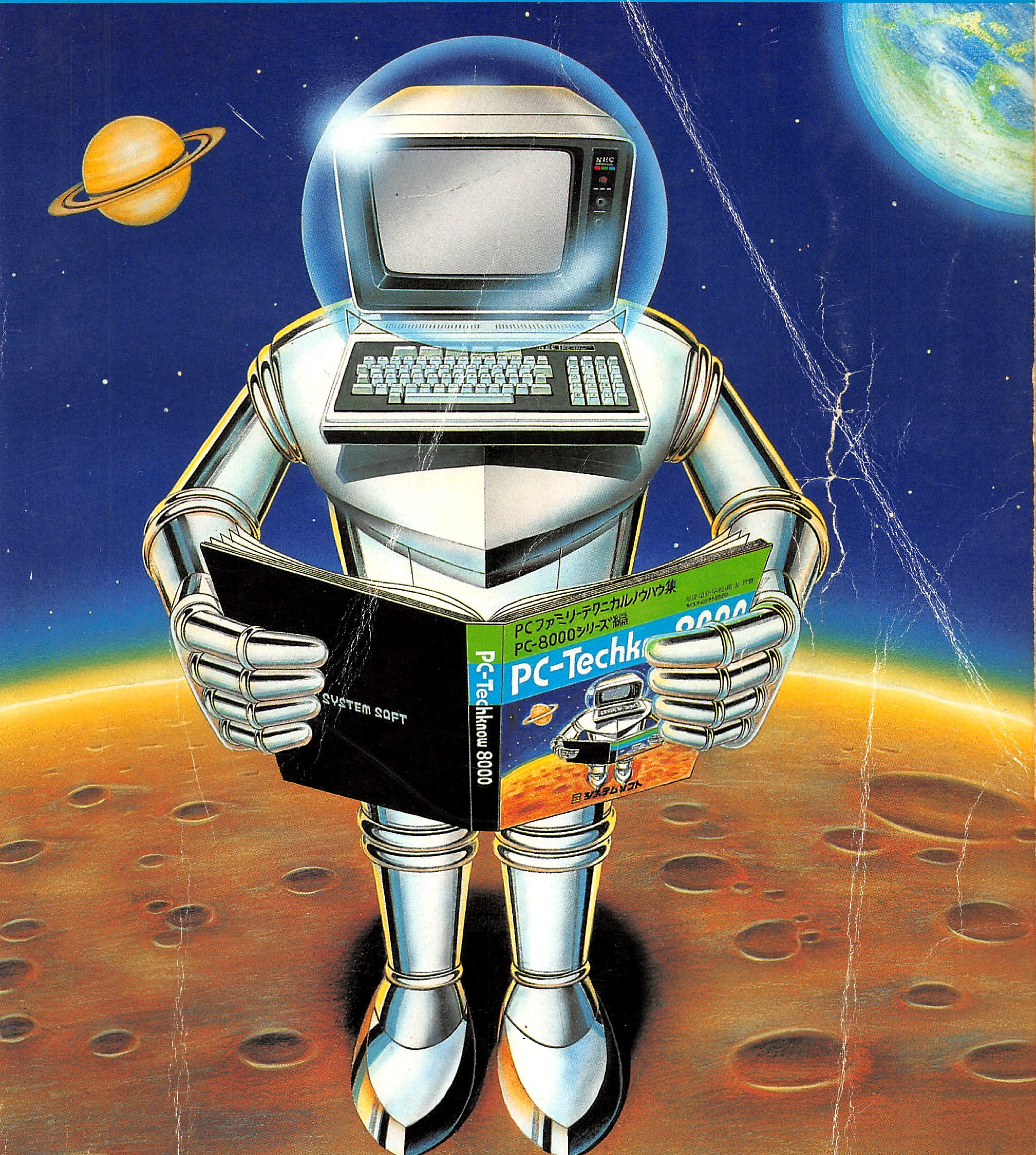
PCファミリー・テクニカル・ノウハウ集
PC-8000シリーズ編

牟田・田川・平松・栗山 共著
システムソフト 監修

PC-Techknow 8000

(テクノウ)



システムソフト

PCファミリー・テクニカル・ノウハウ集
PC-8000シリーズ編

PC-Techknow8000

共著／牟田慎一郎　田川嘉隆
平松達雄　栗山浩一
監修／システム　ソフト

システム ソフト

PCファミリー・テクニカル・ノウハウ集 全3巻の構成

この「ＰＣファミリー・テクニカル・ノウハウ集」は、ＮＥＣのパーソナルコンピュータＰＣファミリー（ＰＣ-8000シリーズ、ＰＣ-6000シリーズ、ＰＣ-8800シリーズ）を徹底的に活用するためのテクニカル・ノウハウ（Tech Know テクノウ）をまとめたもので、以下の全３巻で構成されます。

第１巻　ＰＣ-8000シリーズ編
「ＰＣ-Tech Know 8000」

第２巻　ＰＣ-6000シリーズ編
「ＰＣ-Tech Know 6000」

第３巻　ＰＣ-8800シリーズ編
「ＰＣ-Tech Know 8800」

各巻には、それぞれのBASICの内部構造から、カセットおよびディスク・ファイルをうまく使う方法、機械語理解のポイント、さらにBASICプログラムのテクニックなどが満載されています。本書は、第１巻のPC-8000シリーズ編「PC-Tech Know 8000」であり、第２巻、３巻は順次、発刊予定です。

なお、本書で解説されているN-BASICおよび機械語のプログラムは、BASIC Ver1.0、Ver1.1、Ver1.2(PC-8801)のいずれのバージョンにも適用できます。

はじめに

パーソナルコンピュータが、名実ともに、パーソナルとして誰でも気軽に使えるようになった現在、書店には、その入門書、教本の類があふれ、ＰＣ-8000シリーズに関するものも数多く見受けられます。

しかし、そういう入門書、マニュアルに一通り目を通した人にとって、これらの本に対し、基本的なことしか書かれていないその内容に対するもの足りなさや、数々の疑問が解き明かされないもどかしさを感じられることと思います。

かといって、アセンブラ言語などの機械語をやろうとすると、BASICとのギャップを感じ、思うにまかせないものです。

本書は、こういう方々のために、マニュアルには書かれていないことを中心に、N-BASICやDisk BASICの内部構造、プログラミングのテクニックなど、ＰＣ-8000シリーズを徹底的に活用するためのテクニカル・ノウハウ、さらに、機械語を利用するための手助けとなる内容などを、まとめたものです。

限られたページの中で、充分意を尽すことはできませんが、読者の方々自らによる、内容の補完により、パーソナルコンピュータに対する理解をより深めていただき、その利用技術を高めることに、役立つものと確信します。

パーソナルコンピュータを使いこなすための座右の書として利用していただければ幸いに存じます。

なお、本書の原稿作成に際しては、田川嘉隆氏、平松達雄氏および栗山浩一氏と共同執筆を行ない、システムソフト・スタッフが編集および監修を担当致しました。また、出版については、株式会社システムソフト福岡の樺島正博社長ならびに藤田英時出版部長に大変御世話になりました。ここに厚く御礼申し上げます。

1982年１月

著者代表　牟　田　愼一郎

目　次

目　次

目　次

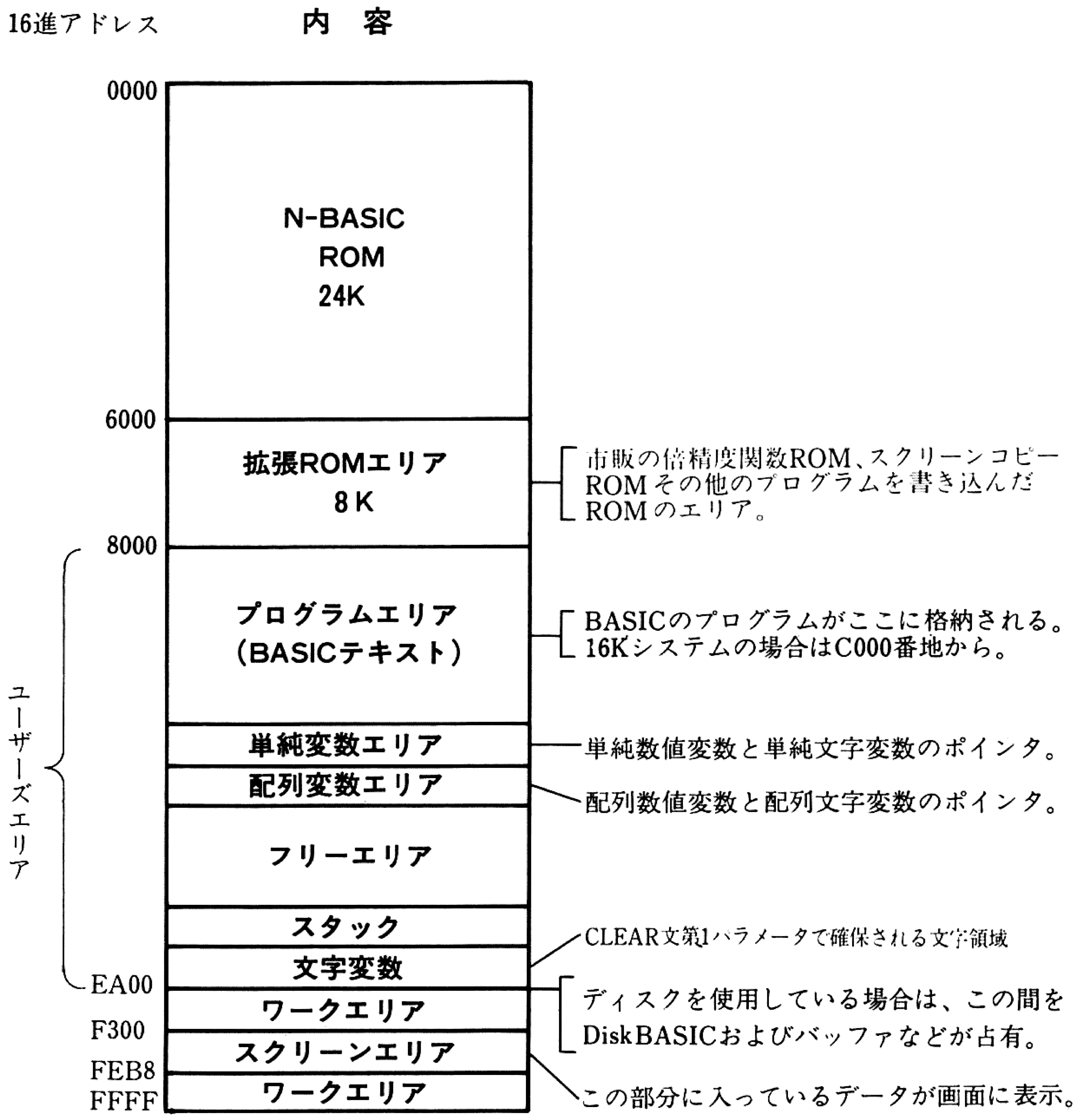
第1章　N-BASICの内部構造

第1章　N-BASICの内部構造

1.メモリマップ

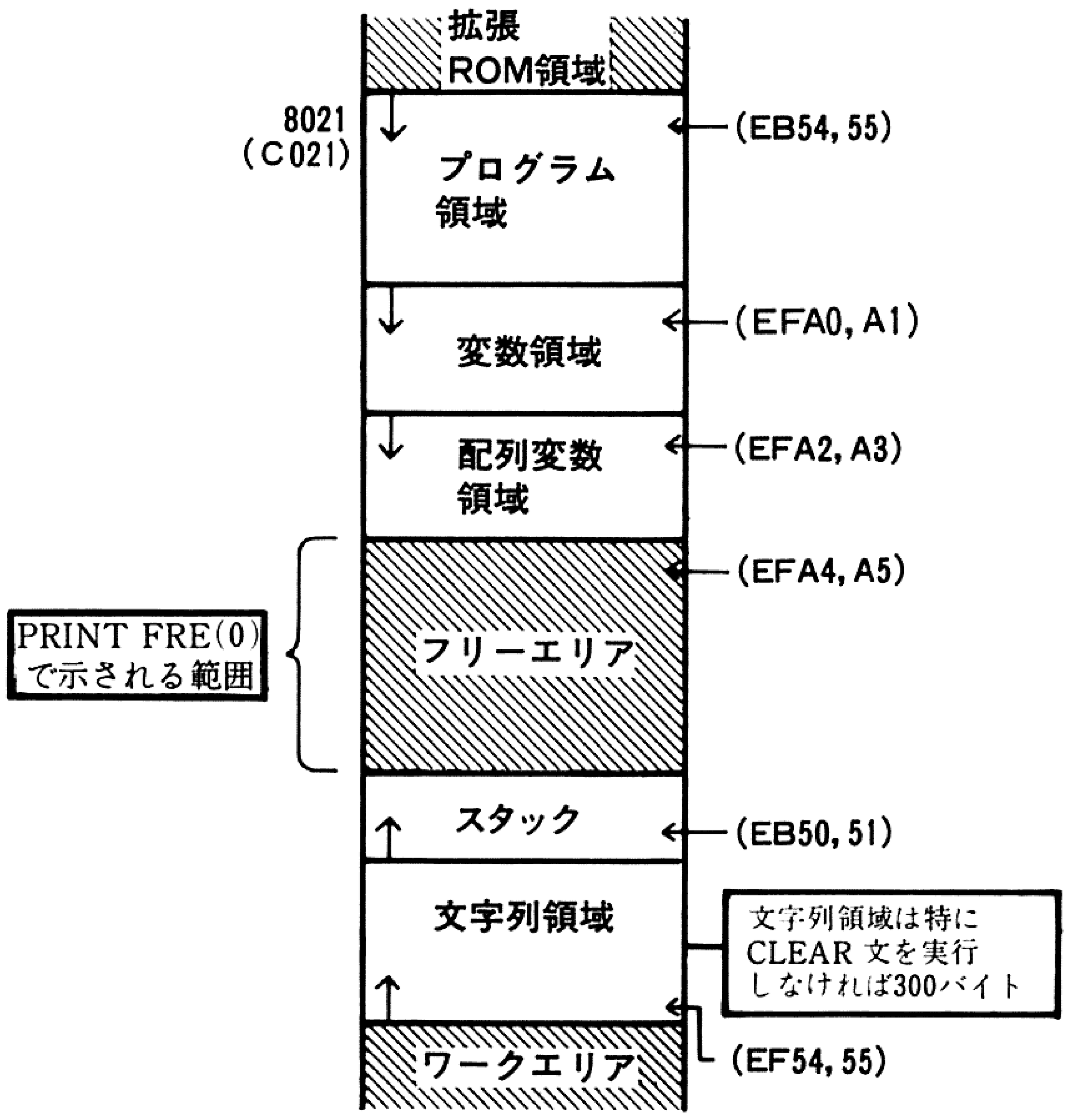
PC−8001内部のメモリ空間は、64Kバイトあり、前半32KバイトがROM、後半32KバイトがRAMに割当てられ、それぞれ下図のようになっています。

〔図表1-1〕メモリマップ



2.ユーザーズエリア

〔図表1－2〕



ユーザーがプログラムを作って実際に使用するエリアは、32Kシステムの場合8000番地からE9FF番地までの約26.5Kバイト、16Kシステムの場合、C000番地からの約10.5Kバイトです。各領域は、右図に示すアドレスにそのポインタが格納されています。

プログラム作成前後、および、実行後についてそのポインタの変化について調べることによってメモリがどのように使われていくかを確認することができます。以後32Kシステムの場合を例にとって説明していきます。

ポインタの内容を調べる方法として、下記の2通りがありますが、一般に1の方法が便利です。

1．モニタモードで“D”コマンドによる方法

```
mon
*DEFA0
EFA0 78 80 86 80 C2 80 20 80 04 04 04 04 04 04 04 04
*
```

2．BASICモードで“peek”による方法

```
print hex$(peek(&heb55)*256+peek(&heb54))
8021
OK
```

実際に下記のサンプルプログラムで電源ON時、プログラム実行前、実行後におけるポインタの変化を調べてみましょう。

```
list
100 REM ｶｹｻﾞﾝ
110 DIM C(12)
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT "AxB=";C(0)
150 END
OK
```

```
run
A= ? 256
B= ? 16
AxB= 4096
OK
```

〔図表１－３〕各ポインタの変化

	ポインタアドレス	電源ON時（RESET）	プログラム実行前	プログラム実行後	
プログラム開始アドレス	EB54,55	8021			プログラム領域
変数領域の始まり	EFA0,A1	8023	8078		単純変数領域
配列変数領域の始まり	EFA2,A3	8023	8078	8086	配列変数領域
フリーエリアの始まり	EFA4,A5	8023	8078	80C2	
スタックの始まり 注）	EB50,51	E8D3			
文字列領域の始まり 注）	EF54,55	E9FF			文字列領域（300バイト）

注)1.CLEAR文を実行すると、このポインタが書き替えられます。
例：CLEAR 1000, &HBFFF 実行後EB50, 51はBC17(BFFF－1000)に、EF54,55はBFFFになります。
2.Disk BASIC起動時は、How many files (0～15)? で指定するFile数により変化します。
例：File数3のときEB50,51はCDEE(Clear300のとき)EF54,55はCF1Aになります。

上の表の内容は、モニタモードで次のようにして調べることができます。

① 電源ON時

```
mon
*DEB50
EB50 D3 E8 FF FF 21 80 20 B7 00 00 00 BB DE DD 00 00
*DEF50
EF50 00 00 00 00 FF E9 58 EF 05 60 3B 00 00 00 00 00
*DEFA0
EFA0 23 80 23 80 23 80 20 80 04 04 04 04 04 04 04 04
*
```

② プログラム実行前

```
mon
¥DEB50
EB50 D3 E8 FF FF 21 80 20 B7 00 00 00 0F 1B 29 22 4C
¥DEF50
EF50 00 00 00 00 FF E9 58 EF 05 60 3B 00 00 00 00 00
¥DEFA0
EFA0 78 80 78 80 78 80 20 80 04 04 04 04 04 04 04 04
¥
```

③ プログラム実行後

```
mon
¥DEB50
EB50 D3 E8 FF FF 21 80 20 B7 00 00 00 0F 1B 29 22 4C
¥DEF50
EF50 00 00 00 00 FF E9 58 EF 05 60 3B 00 00 00 00 00
¥DEFA0
EFA0 78 80 86 80 C2 80 20 80 04 04 04 04 04 04 04 04
¥
```

1-3.プログラムの格納状態

前記サンプルプログラムがメモリ内にどういう形で格納されているか調べてみましょう。

〔図表１－４〕

```
mon               リンクポインタ
¥D8020,80CF       行番号                 中間言語                行の終り リンクポインタ
8020 00 2D 80 64 00 8F 20 B6 B9 BB DE DD 00 39 80 6E
8030 00 86 20 43 28 0F 0C 29 00 47 80 78 00 85 20 22
8040 41 3D 20 22 3B 41 00 5E 80 82 00 85 20 22 42 3D
8050 20 22 3B 42 3A 43 28 11 29 F1 41 F5 42 00 70 80
8060 8C 00 91 20 22 41 78 42 3D 22 3B 43 28 11 29 00
8070 76 80 96 00 81 00 00 00 04 00 41 00 00 00 89 04
8080 00 42 00 00 00 85 04 00 43 37 00 01 0D 00 00 00
8090 00 8D 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

メモリの1つ1つは次のような意味を持っています。

〔図表1-5〕

	アドレス	データ	内容
	8020	00	ここからプログラムが入ります。(EB54,55にそのアドレスが格納されています。)
	8021	2D	802D……リンクポインタ (次の行の開始アドレスを示します。)
	8022	80	
	8023	64	0064……行番号 (10進数に直すと、100となります。)
	8024	00	
100行	8025	8F	REMを意味する中間言語*
100行	8026	20	スペース
100行	8027	B6	〝カ〟
100行	8028	B9	〝ケ〟
100行	8029	BB	〝サ〟
100行	802A	DE	〝〃〟
100行	802B	DD	〝ン〟
	802C	00	行の終わりを示す。
	802D	39	8039……リンクポインタ
	802E	80	
	802F	6E	006E……行番号110
	8030	00	
110行	8031	86	DIMを意味する中間言語
110行	8032	20	スペース
110行	8033	43	〝C〟……変数名
110行	8034	28	(
110行	8035	0F	識別コード…〝0F〟は次が10〜255の整数であることを示します。
110行	8036	0C	10進数で12
110行	8037	29	)
	8038	00	行の終わりを示す。
	8039	47	8047……リンクポインタ
	803A	80	
120行	⋮		
	8047	5E	805E……リンクポインタ
	8048	80	
130行	⋮		
	805E	70	8070……リンクポインタ
	805F	80	
140行	⋮		
	807D	76	8076……リンクポインタ
	8071	80	
150行	⋮		
	8076	00	0000……リンクポインタ
	8077	00	
	8078		ここから変数領域が始まる。 (EFA0,A1にそのアドレスが格納されています。)

リンクポインタが、00・00になったところがプログラムの終わりを示し、その次のアドレス(例では８０７８番地)からが変数領域となります。

1-4.プログラムの回復のさせ方

作ったプログラムが次のような原因でリストも実行もできなくなったり、途中または最後までリストできるが、修正も実行もできない場合があります。

1．NEW を実行したとき。
2．プログラム(とくに機械語を含んだ)が暴走し、RESET せざるを得なくなったかまたは RESET したとき。
3．カセットに保存しようとして、誤って CLOAD を実行したとき。
4．Disk を起動せずに作ったプログラムを、Disk へ入れるため、RESET ボタンを押して Disk を起動したとき。

このような場合、プログラム本体はメモリ内に残っており、次に示すポインタが初期値に戻っているだけです。

最初のリンクポインタ（8021，22番地の内容）⟶00，00
変数領域の始まり　　　（EFA0,A1番地の内容）⟶23，80
配列変数領域の始まり（EFA2,A3番地の内容）⟶23，80
フリーエリアの始まり（EFA4,A5番地の内容）⟶23，80

実際に NEW を実行してメモリ内を見てみましょう。

```
list
100 REM ｶｹｻﾞﾝ
110 DIM C(12)
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT "AxB=";C(0)
150 END
Ok

new
Ok

mon                                                              アドレス
*D8020,807F                                                        802D
8020 00 00 00 64 00 8F 20 B6 B9 BB DE DD 00 39 80 6E
8030 00 86 20 43 28 0F 0C 29 00 47 80 78 00 85 20 22
8040 41 3D 20 22 3B 41 00 5E 80 82 00 85 20 22 42 3D
8050 20 22 3B 42 3A 43 28 11 29 F1 41 F5 42 00 70 80
8060 8C 00 85 20 22 41 78 42 3D 22 3B 43 28 11 29 00
8070 76 80 96 00 81 00 00 00 78 00 00 00 00 00 00 00
*DEFA0                                 アドレス8078
EFA0 23 80 23 80 23 80 20 80 04 04 04 04 04 04 04 04
*
```

□で囲んだ部分が初期値に戻っているのがわかります。

回復させるためには、次のようにします。

1．8021,22番地に次のリンクポインタの先頭アドレス(例では802D)を入れる。

2．EFA0,A1,EFA2,A3,EFA4,A5にそれぞれ最後のリンクポインタ(00,00)の次のアドレス(変数領域の始まるアドレス、例では8078)を入れる。

```
mon
¥S8021
8021 00-2D 00-80 64-
¥SEFA0
EFA0 23-78 80-80 23-78 80-80 23-78 80-80 20-
¥
```

8021,22に入れるべきアドレスは、最初の行の終わりを示す〝00″の次のアドレスであり、更にそのリンクポインタが次のリンクポインタの先頭のアドレスを指しているかどうかで確認します。

また、EFA0～A5に入れるべき変数領域の始まりのアドレスは、行の終わりを示す〝00″とプログラムの終わりを示す〝00,00″のリンクポインタと都合3バイトの〝00″が並んでいる次のアドレスであり、〝00,00″のリンクポインタの先頭を、一つ前のリンクポインタが指しているかどうかで確認します。

以上4ヶ所（8バイト）を書き込んだあとlistすれば、見ごとプログラムが復活しているはずです。

1-5.中間言語

BASICで使用するKEY WORDは、メモリ節約や処理速度向上のため、プログラムエリア内へは、1～2バイトに短縮されたコードすなわち中間言語で記憶されています。

```
list
100 WIDTH 80,20
110 B$=CHR$(12)
120 PRINT B$
OK

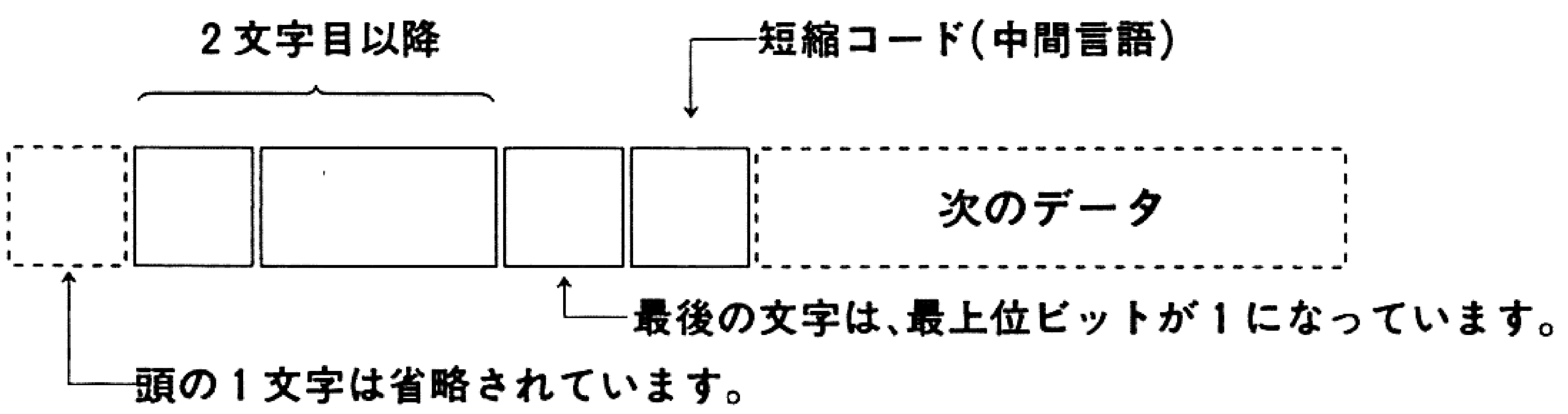
mon
¥D8020,804F
8020 00 2D 80 64 00 A0 20 0F 50 2C 0F 14 00 3B 80 6E
8030 00 42 24 F1 FF 96 28 0F 0C 29 00 44 80 78 00 91
8040 20 42 24 00 00 00 03 00 42 01 FF E9 00 00 00 00
¥
```

このプログラム例では、次のようになっています。

WIDTH	A0
=	F1
CHR$	FF 96
PRINT	91

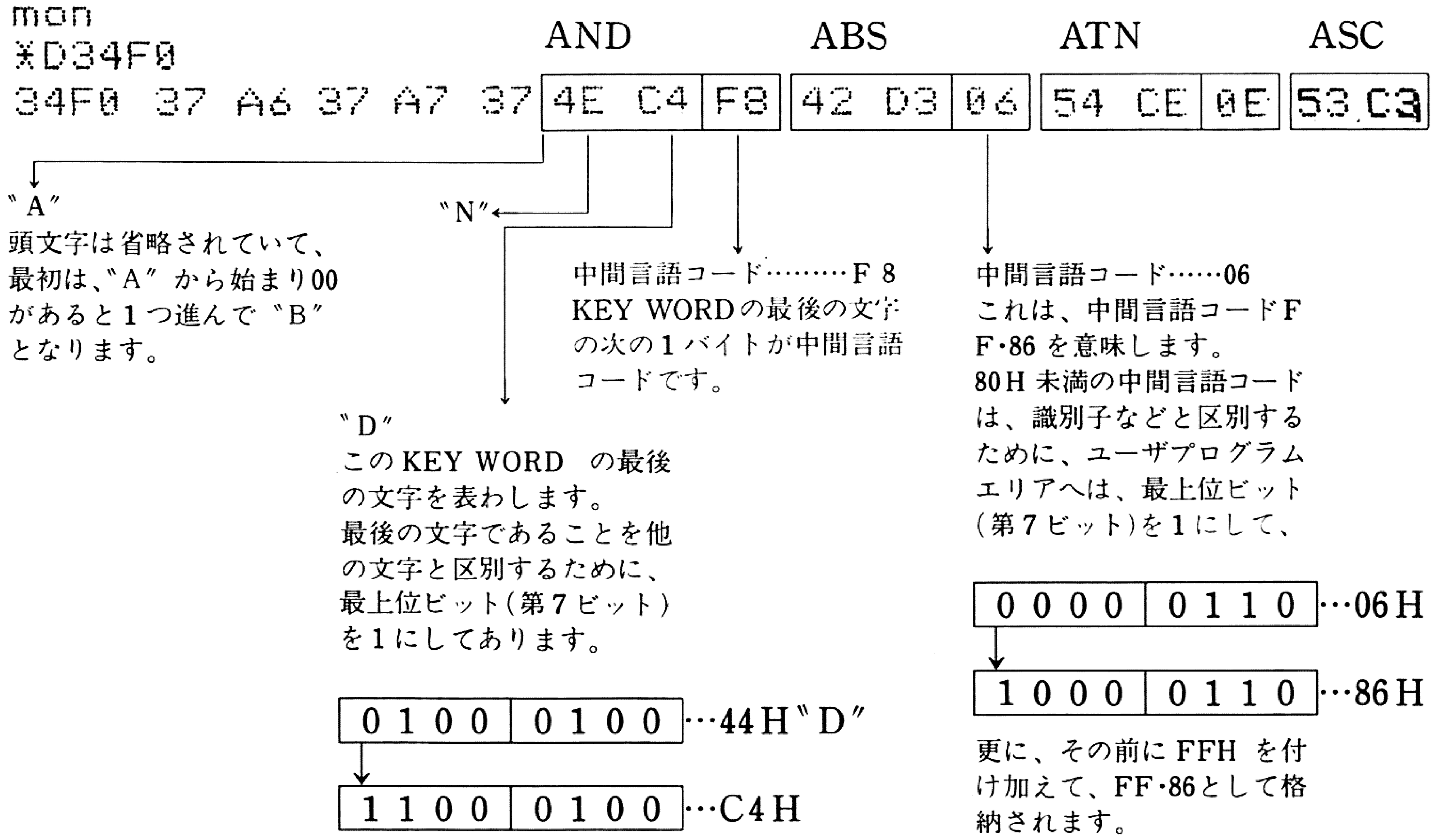
KEY WORD と中間言語の対応表は、BASIC ROM 内の34 F 5番地から37 BB 番地までDATA としてアルファベット順に入っています。

〔図表1-6〕 DATAの形式



実際にダンプリストを見て、調べてみましょう。

〔図表1-7〕



```
            AUTO          ATTR$                  BCD$
3500 [15][55 54 CF A9][54 54 52 A4 E7][00][43 44 A4 1E][45
```

このセパレータにより、以降のKEY WORDは、次の"00"が現れるまで頭文字は"B"となります。

また、この中間言語対応表について、34C1～34F4番地にそのindex部分があります。

```
mon
*D34C0,34FF
34C0 F0 F5 34 0B 35 14 35 5E 35 9D 35 BE 35 DE 35 F0
34D0 35 F5 35 23 36 24 36 2C 36 72 36 9B 36 AA 36 BA
34E0 36 E5 36 E6 36 21 37 5C 37 80 37 89 37 93 37 A2
34F0 37 A6 37 A7 37 4E C4 F8 42 D3 06 54 CE 0E 53 C3
*
```

34C1番地よりABC順に2バイトづつのアドレスが26組格納されています。

表にすると次のようになります。

見方……例えば"G"で始まるKEY WORD(GOTOなど)の中間言語対応データは、35DE番地から始まっていることを表わします。

```
A.....34F5    J.....3623    S.....3721
B.....350B    K.....3624    T.....375C
C.....3514    L.....362C    U.....3780
D.....355E    M.....3672    V.....3789
E.....359D    N.....369B    W.....3793
F.....35BE    O.....36AA    X.....37A2
G.....35DE    P.....36BA    Y.....37A6
H.....35F0    Q.....36E5    Z.....37A7
I.....35F5    R.....36E6
```

上記の表を出力するプログラムは次のとおりです。

```
100 C=ASC("A"):FOR AD=&H34C1 TO &H34F3 STEP 2
110 PRINTCHR$(C)"....."HEX$(PEEK(AD)+PEEK(AD+1)*256)
120 C=C+1:NEXT
```

次のプログラムでKEY WORDと中間言語の対応表を出力してみてください。

```
100 A=ASC("A"):AD=&H34F5
110 KW$=CHR$(A)
120 KW=PEEK(AD):IF KW=0 THEN A=A+1:AD=AD+1:GOTO 110
130 IF KW<&H80 THEN KW$=KW$+CHR$(KW):AD=AD+1:GOTO 120
140 KW$=KW$+CHR$(KW-&H80):AD=AD+1:C=PEEK(AD)
150 IFC<&H80 THEN C$="FF "+HEX$(C+&H80):AC$="・"+CHR$(C+&H80):GOTO 170
160 C$=HEX$(C):IF C<&HF8 THEN AC$=CHR$(C) ELSE AC$="・"
170 PRINT KW$;TAB(8);C$;TAB(15);AC$
180 AD=AD+1:IF AD<&H37BB THEN 110
```

中間言語表（A･B･C順）

AND	F8		・	ELSE	A1		｡	LSET	D0		ﾐ
ABS	FF	86	・▇	ERASE	A5		･	LPRINT	9D		╮
ATN	FF	8E	・▉	ERROR	A6		ｦ	LLIST	9E		╰
ASC	FF	95	・─	ERL	DF		ﾟ	LPOS	FF	98	・┌
AUTO	A9		ｩ	ERR	E0		═	LISTEN	C1		ﾁ
ATTR$	E7		◤	EXP	FF	88	・▏	LIST	93		├
BCD$	FF	9E	・╰	EOF	FF	A7	・ｧ	LFILES	D3		ﾓ
BEEP	B2		ｲ	EQV	F8		・	LOG	FF	8A	・▍
CONSOLE	9F		╯	FORMAT	B3		ｳ	LOC	FF	A8	・ｨ
CLOSE	CA		ﾊ	FOR	82		▃	LEN	FF	92	・┤
CONT	99		┐	FIELD	C6		ﾆ	LEFT$	FF	81	・▂
CLEAR	92		┤	FILES	CD		ﾍ	LOF	FF	A9	・ｩ
CLOAD	9B		┘	FN	DC		ﾜ	MOUNT	C4		ﾄ
CSAVE	9A		└	FRE	FF	8F	・┼	MERGE	CC		ﾌ
CSRLIN	E6		◥	FIX	FF	A2	・｢	MOD	FD		・
CINT	FF	9F	・╯	FPOS	FF	AA	・ｪ	MKI$	FF	AB	・ｫ
CSNG	FF	A0	・	GOTO	89		▎	MKS$	FF	AC	・ｬ
CDBL	FF	A1	・｡	GO TO	89		▎	MKD$	FF	AD	・ｭ
CVI	FF	A3	・｣	GOSUB	8D		▊	MID$	FF	83	・▄
CVS	FF	A4	・､	GET	C7		ﾇ	MOTOR	B9		ｹ
CVD	FF	A5	・･	HEX$	FF	9A	・└	MON	B7		ｷ
COS	FF	8C	・▋	INPUT	85		▆	MAT	C0		ﾀ
CHR$	FF	96	・│	IF	8B		▌	NEXT	83		▄
CMD	B8		ｸ	INSTR	E3		╡	NAME	CE		ﾎ
COLOR	B5		ｵ	INT	FF	85	・▆	NEW	94		▔
DATA	84		▅	INP	FF	90	・┴	NOT	DE		ﾞ
DIM	86		▇	IMP	FC		・	OUT	9C		╭
DEFSTR	AB		ｫ	INIT	D4		ﾔ	ON	95		─
DEFINT	AC		ｬ	INKEY$	E9		♥	OPEN	C5		ﾅ
DEFSNG	AD		ｭ	ISET	BD		ｽ	OR	F9		・
DEFDBL	AE		ｮ	IRESET	BE		ｾ	OCT$	FF	99	・┐
DSKO$	C2		ﾂ	IEEE	FF	EC	・●	PUT	C8		ﾈ
DEF	97		▕	KILL	CF		ﾏ	POKE	98		┌
DELETE	A8		ｨ	KEY	B4		ｴ	PRINT	91		┬
DSKI$	E8		♠	LET	88		▏	POS	FF	91	・┬
DSKF	FF	A6	・ｦ	LOCATE	D5		ﾕ	PEEK	FF	97	・▕
DEC	FF	9D	・╮	LINE	AF		ｯ	PORT	FF	9C	・╭
DATE$	EB		♣	LOAD	CB		ﾋ	POLL	8A		▍
END	81		▂								

PSET	B1	ア	SPC(	DD	ン	USING	E2	╪
PRESET	B0	ー	STEP	DA	レ	USR	DB	ロ
POINT	EF	╲	SGN	FF 84	.▄	VAL	FF 94	.‾
READ	87	█	SQR	FF 87	.█	VARPTR	E5	◣
RUN	8A	▌	SIN	FF 89	.▌	WIDTH	A0	
RESTORE	8C	▌	STR$	FF 93	.├	WAIT	96	╎
RETURN	8E	█	STRING$	E1	╞	WBYTE	BC	シ
REMOVE	C3	テ	SPACE$	FF 98	.┌	XOR	FA	・
REM	8F	┼	STATUS	EE	╱	+	F3	月
RESUME	A7	ァ	SRQ	ED	○	−	F4	日
RSET	D1	ム	TRON	A2	「	＊	F5	時
RIGHT$	FF 82	.▄	TROFF	A3	」	/	F6	分
RND	FF 88	.▏	TAB(	D9	ル	^	F7	秒
RENUM	AA	ェ	TO	D7	ラ	¥	FE	・
RBYTE	BB	サ	THEN	D8	リ	'	E4	◢
STOP	90	┴	TAN	FF 8D	.▌	>	F0	╳
SWAP	A4	、	TERM	B6	カ	=	F1	円
SET	C9	ノ	TALK	BF	ソ	<	F2	年
SAVE	D2	メ	TIME$	EA	◆			

注）中間言語に対するキャラクタはKEY WORDを探す際に便利なように併せて、出力しています。

〔図表1-8〕中間言語表（コード順）

下＼上	8	9	A	B	C	D	E	F
0		STOP	WIDTH	ー PRESET	タ MAT	ミ LSET	ERR	╳ >
1	END	PRINT	。 ELSE	ア PSET	チ LISTEN	ム RSET	STRING$	円 =
2	FOR	CLEAR	「 TRON	イ BEEP	ツ DSKO$	メ SAVE	USING	年 <
3	NEXT	LIST	」 TROFF	ウ FORMAT	テ REMOVE	モ LFILES	INSTR	月 +
4	DATA	NEW	、 SWAP	エ KEY	ト MOUNT	ヤ INIT	◢ '	日 −
5	INPUT	ON	・ ERASE	オ COLOR	ナ OPEN	ユ LOCATE	◣ VARPTR	時 ＊
6	DIM	WAIT	ヲ ERROR	カ TERM	ニ FIELD	ヨ	◥ CSRLIN	分 /
7	READ	DEF	ァ RESUME	キ MON	ヌ GET	ラ TO	◤ ATTR$	秒 ^
8	LET	POKE	ィ DELETE	ク CMD	ネ PUT	リ THEN	♠ DSKI$	AND
9	GOTO	CONT	ゥ AUTO	ケ MOTOR	ノ SET	ル TAB(	♥ INKEY$	OR
A	RUN	CSAVE	ェ RENUM	コ POLL	ハ CLOSE	レ STEP	◆ TIME$	XOR
B	IF	CLOAD	ォ DEFSTR	サ RBYTE	ヒ LOAD	ロ USR	♣ DATE$	EQV
C	RESTORE	OUT	ャ DEFINT	シ WBYTE	フ MERGE	ワ FN	●	IMP
D	GOSUB	LPRINT	ュ DEFSNG	ス ISET	ヘ FILES	ン SPC(	○ SQR	MOD
E	RETURN	LLIST	ョ DEFDBL	セ IRESET	ホ NAME	゛ NOT	╱ STATUS	¥
F	REM	CONSOLE	ッ LINE	ソ TALK	マ KILL	゜ ERL	╲ POINT	＊

（FF＋　）

8	9	A
	INP	CSNG
LEFT$	POS	CDBL
RIGHT$	LEN	FIX
MID$	STR$	CVI
SGN	VAL	CVS
INT	ASC	CVD
ABS	CHR$	DSKF
SQR	PEEK	EOF
RND	SPACE$	LOC
SIN	OCT$	LOF
LOG	HEX$	FPOS
EXP	LPOS	MKI$
COS	PORT	MKS$
TAN	DEC	MKD$
ATN	BCD$	
FRE	CINT	

＊FFについては、上図を参照。
＊IEEEの中間言語は、FF・ECです。

中間言語表にキャラクタを一緒に出力しておりますが、これは、1−12に示すダンプリスト(アスキー文字同時出力)でテキストを調べる場合、便利ですので活用してください。

例えば、PRINTは、中間言語コードで91Hなので、アスキーダンプ部分には、“┬”で出力されています。次のサンプル・プログラムとそのダンプリストを参考にしてください。

BASICプログラム

```
100 PRINT "SYSTEM"
110 PRINT "SOFT"
120 PRINT "FUKUOKA"
130 PRINT "PHONE:092-714-6254"
```

ダンプリスト

```
                                                                          アスキーダンプ部
8020  00 30 80 64 00 91 20 22  53 59 53 54 45 4D 22 00  3EE  ·0_d·┬ " SYSTEM"·
8030  3D 80 6E 00 91 20 22 53  4F 46 54 22 00 4D 80 78  4A1  =_n·┬ "S OFT"·M_x
8040  00 91 20 22 46 55 4B 55  4F 4B 41 22 00 68 80 82  475  ·┬ "FUKU OKA"·h_▬
8050  00 91 20 22 50 48 4F 4E  45 3A 30 39 32 2D 37 31  3B7  ·┬ "PHON E:092-71
8060  34 2D 36 32 35 34 22 00  00 00 FF 00 00 00 00 00  253  4-6254"· ........
8070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  000  ........ ........
8080  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF0  ........ ........
```

1-6.中間言語処理ルーチン

中間言語に対応する処理ルーチンのアドレスが、33BD番地から34C0番地に中間言語コード順に格納されています。

```
33BDH     81H..............END
  ∫        ∫
3466H     D5H.............LOCATE

3467H     FF81H..........LEFT$
  ∫         ∫
34C0H     FFADH.........MKD$
```

N-BASICで使用するKEY WORDの場合は、その処理ルーチンは、直接N-BASICインタプリタ内（0〜5FFF）を指していますが、Disk BASICで使用するKEY WORDの場合、一旦F0E1〜F215番地にあるジャンプテーブルを呼び、Disk BASICが起動している場合には、そこから、Disk BASICインタプリタ内（D3E5〜E9FF）へジャンプし、そうでない場合は、1875番地へジャンプします。

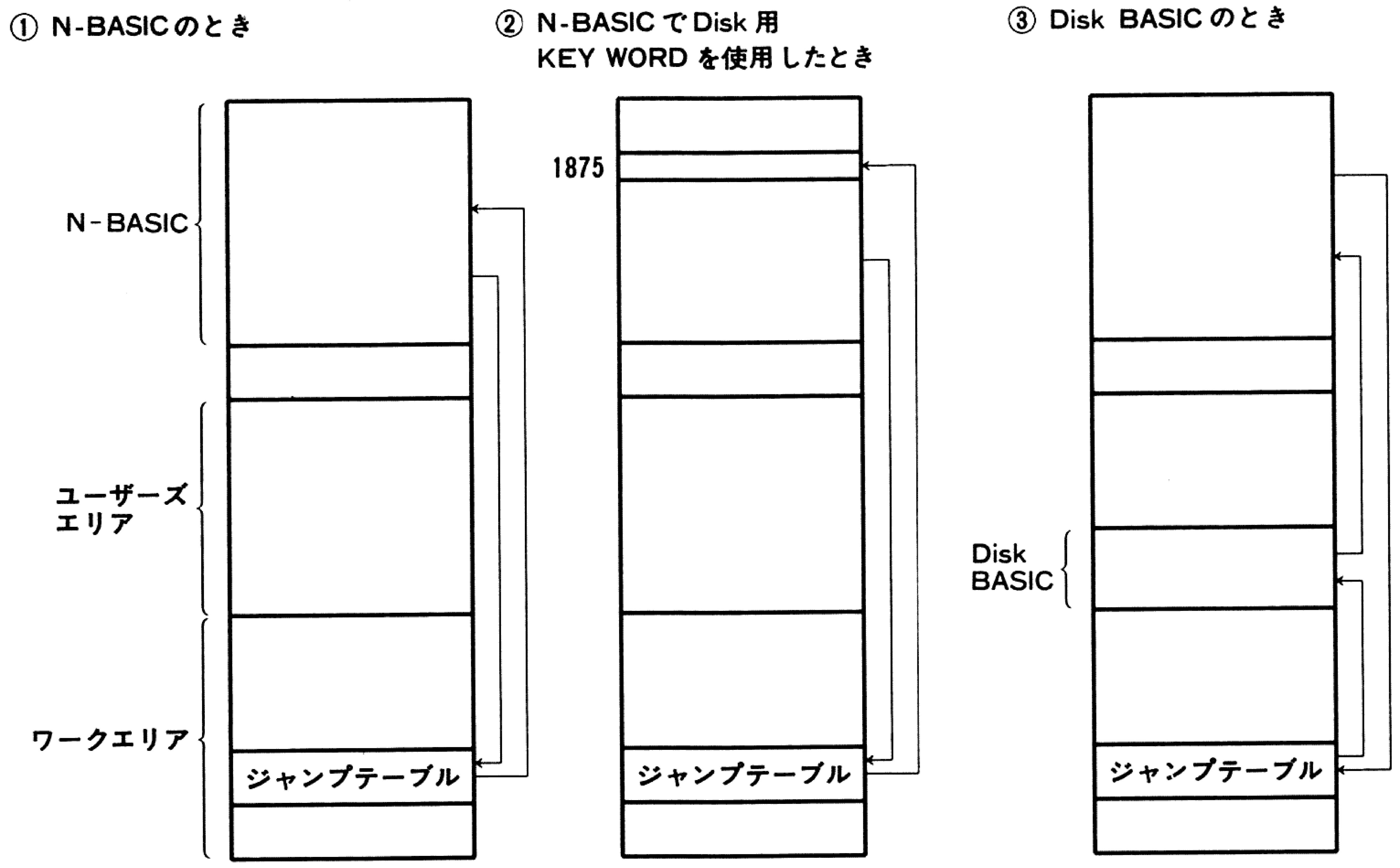
1875番地は、エラーメッセージ〝Disk BASIC Feature〟を出力して、ダイレクトコマンドモードに戻るルーチンです。

MON モードでジャンプして実行すると、

```
mon
*G1875
Disk BASIC Feature
OK
```

となります。

〔図表1-9〕



ジャンプテーブルは、各3バイトで、未使用分も含めて、全部で85組あります。

N-BASICのみの場合、Disk BASIC用KEY WORDに対応するジャンプテーブル内はC3・75・18で1875番地へジャンプする機械語が入っており、Disk BASIC Featureを表示してダイレクトコマンドモードに戻ります。

Disk BASICの場合、そのジャンプテーブルは、起動時に、Disk BASICインタプリタ内のジャンプテーブル（D58B～D6BF）が、F0E1～F215番地にブロック転送され、Disk BASIC用に書き換えられてしまい、先程のC3・75・18が、例えばC3・84・DBなどとなり、DB84番地にジャンプすることになります。

したがって、Disk BASIC KEY WORDを使用すると、それに対応したDisk BASICインタプリタ内の処理ルーチンへジャンプすることがわかります。

次のプログラムで、中間言語コード対応の処理ルーチンのリストが出力できます。

```
100 ' *****   Subroutine for Keywords   *****
110 C=&H81:C$="":FOR AD=&H33BD TO &H34BF STEP 2
120 IF AD=&H3467 THEN C=&H81:C$="FF "
130 A=PEEK(AD)+PEEK(AD+1)*256
140 PRINT C$;HEX$(C);TAB(7);RIGHT$("000"+HEX$(A),4)"  ";
150 IF A<&H7FFF THEN PRINT:GOTO 170
160 A=PEEK(A+1)+PEEK(A+2)*256:PRINT RIGHT$("000"+HEX$(A),4)
170 C=C+1:NEXT
```

実行結果を次に示します。

81	432F	A6	46C4	CB	F138 DDF0
82	4159	A7	468C	CC	F13B DDF1
83	4A08	A8	58D9	CD	F14D E284
84	45BE	A9	46CF	CE	F13E DB84
85	48DA	AA	5AED	CF	F141 DDDE
86	4E37	AB	445B	D0	F144 E012
87	4939	AC	445E	D1	F147 E011
88	45DE	AD	4461	D2	F14A DE3D
89	456D	AE	4464	D3	F126 E27F
8A	453D	AF	4877	D4	2262
8B	4702	B0	0705	D5	0792
8C	4302	B1	06B8	FF 81	54F9
8D	4555	B2	0D41	FF 82	5529
8E	45A3	B3	F12F D804	FF 83	5532
8F	45C0	B4	1343	FF 84	2686
90	432A	B5	0951	FF 85	283F
91	4742	B6	0DB8	FF 86	2671
92	44E8	B7	0D5D	FF 87	31A1
93	570C	B8	F0FC 1875	FF 88	3283
94	3DE0	B9	0DA1	FF 89	32FC
95	4642	BA	F114 1875	FF 8A	2503
96	56B3	BB	F11A 1875	FF 8B	31F3
97	50CC	BC	F117 1875	FF 8C	32F6
98	5918	BD	F105 1875	FF 8D	335D
99	4383	BE	F108 1875	FF 8E	3372
9A	1EC0	BF	F10B 1875	FF 8F	5051
9B	1F10	C0	F111 1875	FF 90	56A1
9C	56AD	C1	F10E 1875	FF 91	5079
9D	473A	C2	F12C E47C	FF 92	548C
9E	5707	C3	F150 E20D	FF 93	527A
9F	0884	C4	F153 E208	FF 94	5553
A0	0843	C5	F0FF DBD6	FF 95	5498
A1	45C0	C6	F102 E0BF	FF 96	54A8
A2	4396	C7	1886	FF 97	5911
A3	4397	C8	1891	FF 98	54DF
A4	439C	C9	F156 E109	FF 99	5270
A5	43DF	CA	F135 E240	FF 9A	5275

```
FF 9B  5074          FF A2  282C          FF A9  F0F0 E83B
FF 9C  20F9          FF A3  F0E1 E1EB     FF AA  F168 E868
FF 9D  F123 1875     FF A4  F0E4 E1EE     FF AB  F0F3 E1D2
FF 9E  F120 1875     FF A5  F0E7 E1F1     FF AC  F0F6 E1D5
FF 9F  277F          FF A6  F159 E812     FF AD  F0F9 E1D8
FF A0  27B3          FF A7  F0EA E84C
FF A1  27DF          FF A8  F0ED E823
```

注1）中間言語表より、例えば81Hは END、B9Hは MOTOR、FF・96は、CHR$の中間コードです。

注2）N-BASICのみの場合、2つめの処理ルーチンアドレスは、すべて、1875になります。

なお、付録のKEY WORD、中間言語コードおよび処理ルーチン対応表も参照してください。

この出力結果を見てみると、Disk BASIC Featureのエラーメッセージを出すルーチン（1875H番地）へジャンプしているものがいくつかあることがわかります。

中間言語コード	KEY WORD	ジャンプテーブルアドレス
B8	CMD	F0FC
BA	POLL	F114
BB	RBYTE	F11A
BC	WBYTE	F117
BD	ISET	F105
BE	IRESET	F108
BF	TALK	F10B
C0	MAT	F111
C1	LISTEN	F10E
FF・9D	DEC	F123
FF・9E	BCD$	F120

これらのKEY WORDは、現在のシステムでは使われていないことになるので、空き領域や、CLEAR文で確保した領域に、機械語のルーチンをつくり、そのアドレスを対応するジャンプテーブルに、C3・[下位]・[上位]と書込んでおけば、これらのKEY WORDにより、そのルーチンが実行されます。7-4-4～5にその応用例をあげておりますので参照してください。

このようにして使われているジャンプテーブルは、85組のうちの39組で、これ以外にもN-BASICやDisk BASICの要所要所からこのテーブルに飛んできては、別のところへジャンプしていったり（C3・5D・E1）、すぐさま元のところに戻っていったり（C9・00・00）していますので、機械語にくわしい方は、これらも利用すると、更に応用が広がります。

1-7.識別コード

識別コードは、プログラム中で、数値の先頭につけられていて、以下の数値が、何であるかを示すものであり、KEY WORDや変数名との区別を行なうものです。

次表に、識別コードとその意味について、示します。

〔図表1-10〕

識別コード	意味
0B	(&O) 以下の2バイトは、プログラム中の8進数を16進数に変換したものです。(−32768〜32767…整数)
0C	(&H) 以下の2バイトは、16進数です。(−32768〜32767…整数)
0D	以下の2バイトは、行番号などを、実際のメモリ上の飛び先アドレス(16進)に変換したものです。
0E	以下の2バイトは、GOTO, GOSUB, THENの後に続く、行番号(0〜65535)を16進に変換したものです。 注) 0Eの後の行番号は、一度プログラムを実行すると、実際のアドレスに変換され、識別コードも"0D"に変わります。
0F	以下の1バイトは、10〜255の数値を16進で表わしたものです。(0A〜FF)
11〜1A	識別コードそのものが0〜9の数値を表わします。(11→0, 12→1, 13→2, ..., 19→8, 1A→9)
1C	以下の2バイトは、整数(−32768〜32767)を16進数に変換したものです。
1D	以下の4バイトは、浮動小数点表記の単精度実数です。
1F	以下の8バイトは、浮動小数点表記の倍精度実数です。

1-8.単純変数領域

単純変数領域は、プログラム領域のすぐ後から、配列変数領域の前までで、その領域はポインタEFA0, A1で示されるアドレスからEFA2, A3で示されるアドレスの1つ前までです。但し、プログラムを実行して実際にその変数が使われないと、変数領域には登録されません。またプログラムを変更すると自動的にクリアされます。

```
list
100 REM ｶｹｻﾞﾝ
110 DIM C(12)
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT "AxB=";C(0)
150 END
Ok

run
A= ? 256
B= ? 16
AxB= 4096
Ok

mon
*DEFA0
EFA0 78 80 86 80 C2 80 20 80 04 04 04 04 04 04 04 04
*
```

（EFA0の「78 80」→ 変数領域の始まりアドレス、「86 80」→ 配列変数領域の始まりアドレス）

上の例では、単純変数領域のアドレスは、8078～8085番地ということになります。
では、その中味はどうなっているか調べてみましょう。

```
*D8070,808F
8070 76 80 96 00 81 00 00 00 04 00 41 00 00 00 89 04
8080 00 42 00 00 00 85 04 00 43 37 00 01 0D 00 00 00
*
```

（8078～8085 → 単純変数領域）

8078　0 4　　変数の型──→（1バイト）

0 2	整数型変数
0 3	文字型変数
0 4	単精度型変数
0 8	倍精度型変数

または、変数名の後に占有するバイト数とみても良いでしょう。

8079　0 0
7A　4 1　　変数名　変数名A（アスキーコード41H）（2バイト）
変数名が2文字の場合、2文字目が1バイト目に入ります。

VARPTR関数はこの→807B　0 0
アドレスを指します　～　0 0
　0 0
7E　8 9　　数値256を表わしています。（浮動小数点表記による）
くわしくは、8－13を参照してください。

807F～85　　変数Bの内容（上記に同じ）

〔図表1-11〕

	変数の型	変数名	数値またはディスクリプタ部分	合 計
整数 変数	02	2バイト	2バイト	5バイト
単精度変数	04	2バイト	4バイト	7バイト
倍精度変数	08	2バイト	8バイト	11バイト
文字 変数	03	2バイト	文字数 / ポインタ*	6バイト

↓──VARPTR関数は、ここのアドレスを指します。（数値またはディスクリプタ部分の先頭）

ストリングディスクリプタ（文字数＋ポインタ*）

ポインタ：文字列の格納先（プログラム中または、文字列領域）アドレスを示します。

変数の格納アドレスを知るのにVARPTRという関数があります。

```
print hex$(varptr(a))
807B
OK
print hex$(varptr(c(0)))
808E
OK
```

この関数により示されるアドレスは、変数名の次のアドレス、すなわち数値変数の場合数値部分の先頭アドレスを、文字変数の場合、ストリングディスクリプタの先頭アドレス(文字数の部分)を指しています。

1-9.配列変数領域

配列変数領域は、単純変数領域の後にありフリーエリアの始まる前までで､その領域は、ポインタEFA2,A3で示されるアドレスからEFA4,A5で示されるアドレスの1つ前までです。この領域は、DIM文を実行するか､添字が10以下の配列変数を使ったときに確保されます。配列変数領域が確保されたあとに新たな単純変数を使うと、配列変数領域は、その分だけ後にずれます。また単純変数領域と同様、プログラムを変更すると自動的にクリアされます。

```
list
100 REM ｶｹｻﾞﾝ
110 DIM C(12)
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT "AxB=";C(0)
150 END
OK

run
A= ? 256
B= ? 16
AxB= 4096
OK

mon
*DEFA0
EFA0 78 80 86 80 C2 80 20 80 04 04 04 04 04 04 04 04
*
```

（上の出力で「86 80」に「配列変数領域の始まり」、「C2 80」に「フリーエリアの始まり」の注記あり）

上の例では、8086～80C1番地が配列変数領域ということがわかります。その領域をはっきりするために、C(12)に適当な値(ここでは12.3) を代入してみると、

```
C(12)=12.3
OK

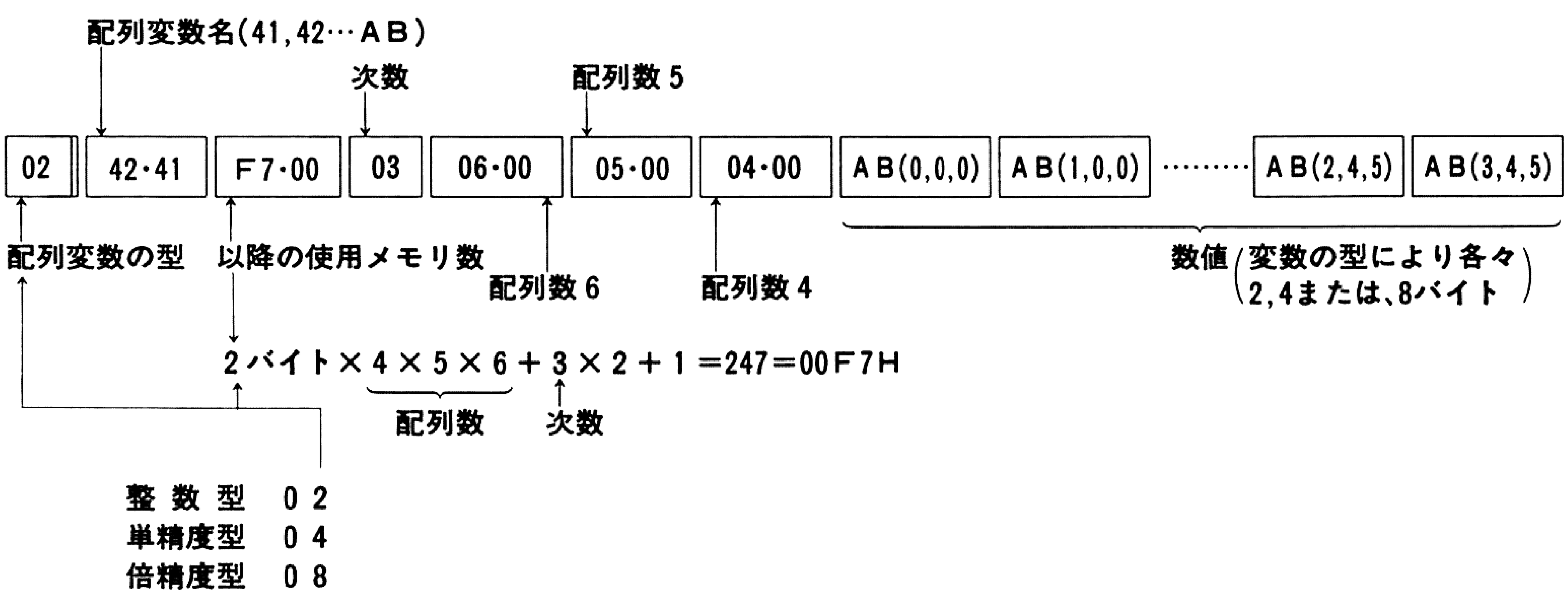
mon
*D8080,80CF
8080 00 42 00 00 00 85 04 00 43 37 00 01 0D 00 00 00
8090 00 8D 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CD CC
80C0 44 84 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
```

（8086～80C1番地が「配列変数領域」として囲まれている）

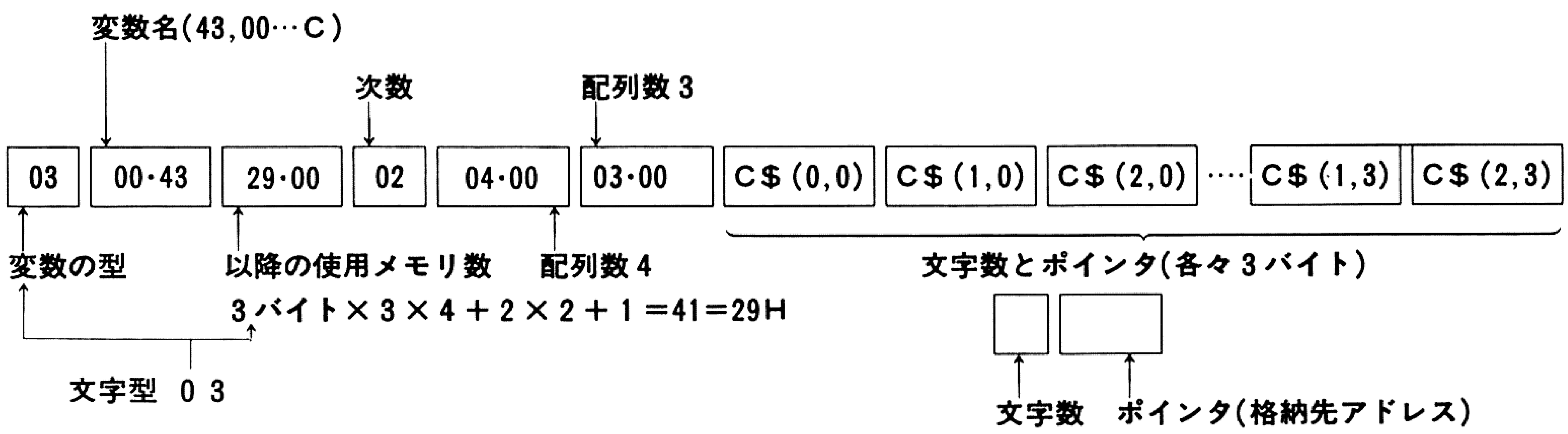
```
8086  04………配列変数の型
8087  00 )
  88  43 )……43・00  配列変数名〝C″…アスキーコード…43H
8089  37 )
8 8A  00 )……00・37  以降に使用するメモリ数(4バイト×13＋3＝56→37H)
808B  01………次元数  1次元配列を表わす。
808C  0D )
  8D  00 )……00・0D  配列の要素大きさ  13…(添字の数＋1)
808E  00 )
      00 )
  ∫   00 )……数値  C(0)の数値  ┐
  91  8D )                        │
  :    :                          ├ 4×13＝52バイト
80BE  CD )                        │
      CC )                        │
  ∫   44 )……数値  C(12)の数値 ┘
  C1  84 )
```

数値配列と文字配列について、整理すると、次のようになります。

〔図表1-12〕数値配列の場合　AB%(3, 4, 5)の例



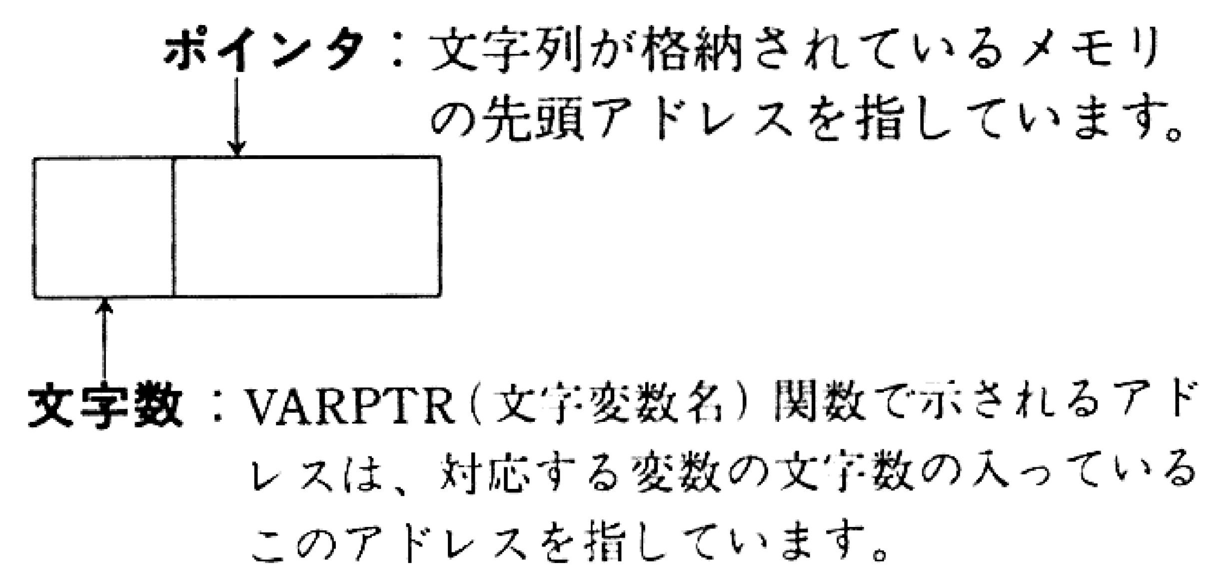
〔図表1-13〕文字配列の場合　C$(2, 3)の例



1-10.文字変数と文字列領域

変数が文字列の場合、単純変数領域や配列変数領域に格納されているのは文字データそのものでなく、変数の型(1バイト……03H)、変数名(2バイト)、配列変数の場合更にメモリ数(2バイト)、次元数(1バイト)、配列数(2バイト×次数)と、それに続く、ストリングディスクリプタ(3バイト)と呼ばれるものだけです。

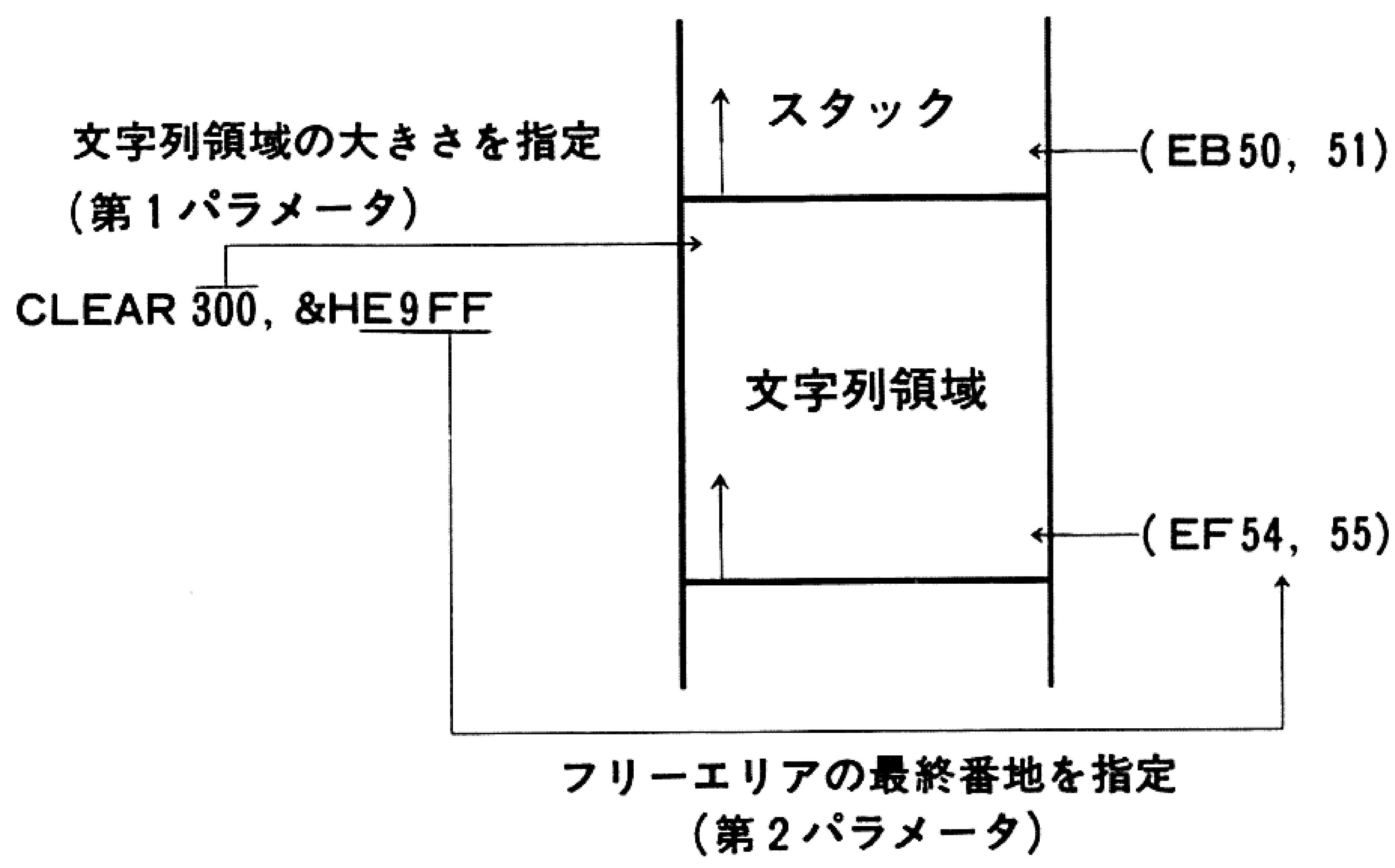
〔図表1-14〕ストリングディスクリプタの構成



実際の文字列は、このストリングディスクリプタ中のポインタの指すアドレスから始まっており、その場所は、プログラムエリア中や文字列領域中、また、Disk BASICの場合は、ファイルバッファ中などとなっています。

文字列領域は、EB50, 51で示されるアドレス+1からEF54, 55で示されるアドレスまでです。

〔図表1-15〕



スタックの始まりアドレス(EB50, 51に格納)は、フリーエリアの最終アドレス(EF54, 55に格納)からCLEAR文の第1パラメータの数値を差引いたものに等しくなります。

次のプログラム例で確認してみましょう。

```
list
100 A$="PC-8001"
110 FOR I=0 TO 2:READ NM$(I):NEXT
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
OK
```

プログラムの実行後、変数のポインタ（ストリングディスクリプタのアドレス）を調べてみましょう。

```
run
Phone No. ? 092-714-6254
OK
```

```
print hex$(varptr(a$)),hex$(varptr(ph$)),hex$(varptr(nm$(0)))
8081          808E          8099
OK
```

次に、上記ポインタのアドレスを調べてみます。

```
mon
*D8080,809F
8080 41 07 29 80 04 00 49 00 00 40 82 03 48 50 0C F4
8090 E9 03 4D 4E 24 00 01 0B 00 06 51 80 04 58 80 07
*
```

〔図表1-16〕

変数名	ストリングディスクリプタのアドレス	ストリングディスクリプタ		文字列格納場所
		文字数	ポインタ	
A$	8081	7	8029	プログラム領域
PH$	808E	12	E9F4	文字列領域
NM$(0)	8099	6	8051	プログラム領域

文字列領域は、EB50, 51と、EF54, 55を調べると、

```
mon
*DEB50         ┌スタックポインタ初期アドレス
EB50 [D3 E8] FF FF 21 80 20 B7 00 00 00 28 0F 28 1B 24
*DEF50
EF50 00 00 00 00 [FF E9] 58 EF 05 60 3B 00 00 00 00 00
*                     └フリーエリア最終アドレス
```

E8D4～E9FFがその領域であることがわかります。したがって、PH$のE9F4は、文字列領域の終わりの方にあることになります。

```
mon
*D8E9F0,E9FF
E9F0 00 00 00 00 30 39 32 2D 37 31 34 2D 36 32 33 36
*
```

すなわち、文字列は、文字領域の終わりの方から格納されます。また、A$, NM$(I)のように、プログラム中に代入文やDATA文などで文字列が存在する場合は、文字領域は使用されません。

注）Field文で指定した変数のポインタは Disk のファイルバッファ内を指しています。

1-11.フリーエリア

フリーエリアの量を知る関数として、fre関数があり、(　)内に入れる引数が数値または数値変数の場合は、BASICプログラム領域の空きエリアのバイト数を、引数が文字または文字変数の場合は、文字列スタック領域の空きエリアのバイト数を、それぞれ表わします。

N-BASICの起動時、32Kシステムでは、

```
NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Microsoft

Ok
print fre(0)
 26786
Ok
print fre(a$)
 300
Ok
```

16Kシステムの場合

```
print fre(0)
 10402
Ok
```

となり、26786バイトのテキストエリアと300バイトの文字列スタック領域があることがわかります。

文字列スタック領域の300バイトは、CLEAR文により自由に容量変更できます。

Disk BASIC起動時は、Disk BASICプログラムおよびファイルバッファ領域などが占有されて、例えば、ファイルバッファを3だけ確保した場合には、32Kシステムの場合、

```
Disk version [14-Nov-1980]
How many files(0-15)? 3
NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Microsoft

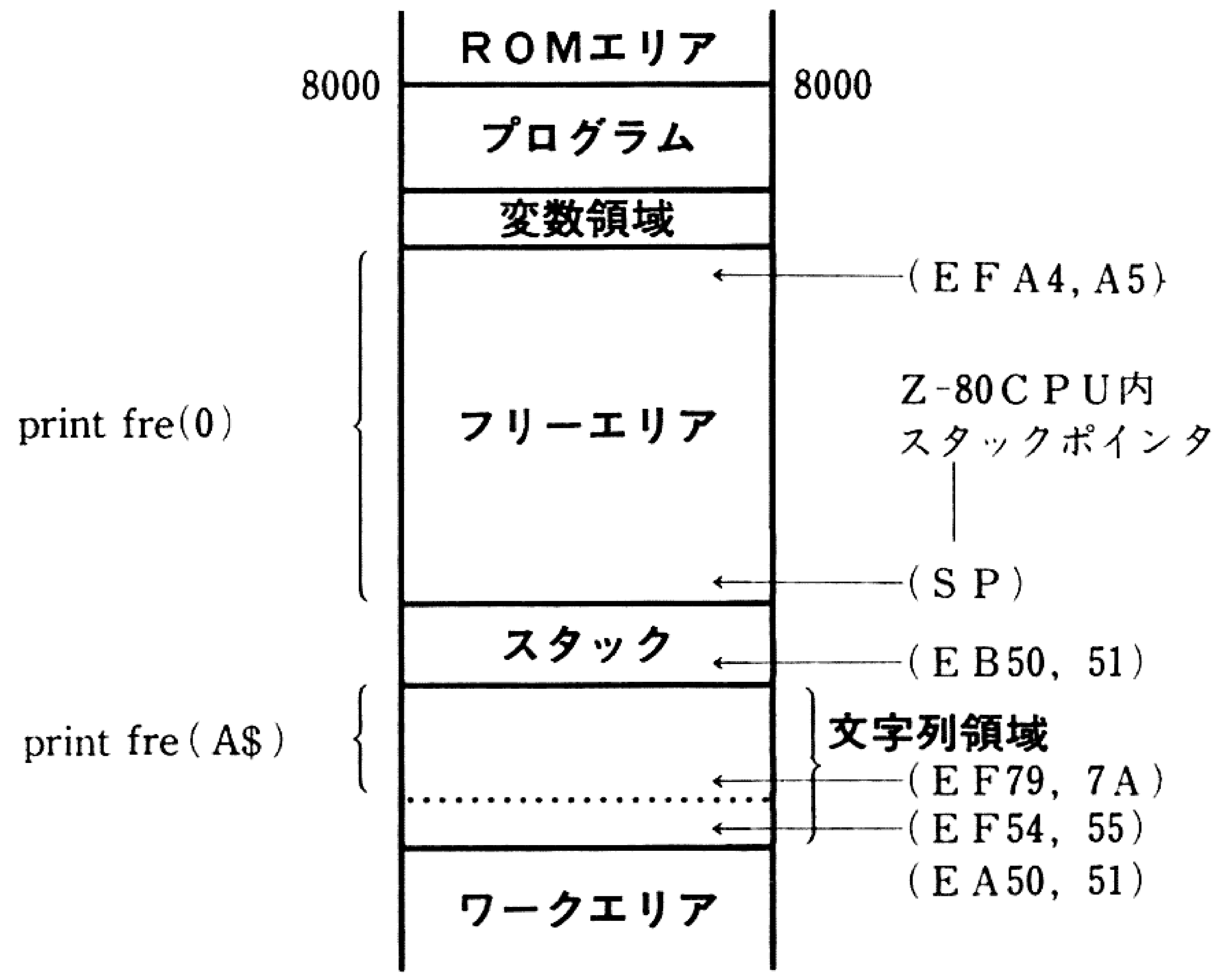
Ok
print fre(0)
 19901
Ok
```

となり、前と比較すると、26786−19901＝6885と、約７Kバイト弱、プログラムエリアが少なくなっていることがわかります。

フリーエリアに関するアドレスは、次の通りです。

EFA4，A5…… フリーエリアの先頭アドレス
EB50，51……… スタックの始まりアドレス
EF79，7A……… ストリング(文字列)フリースペース先頭アドレス
EF54，55……… フリーエリアの最終アドレス(CLEAR文で変わる)
EA50，51……… フリーエリアの最終アドレスの初期値(N-BASIC, Disk BASICにおいても常にE9FFでCLEAR文を実行しても変化しません。)

〔図表1-17〕



もう一度、次のプログラムをキーイン・実行して、フリーエリアのアドレスの変化を調べてみましょう。

(32Kシステム　N-BASIC起動時)

```
list
100 A$="PC-8001"
110 FOR I=0 TO 2:READ NM$(I):NEXT
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
Ok

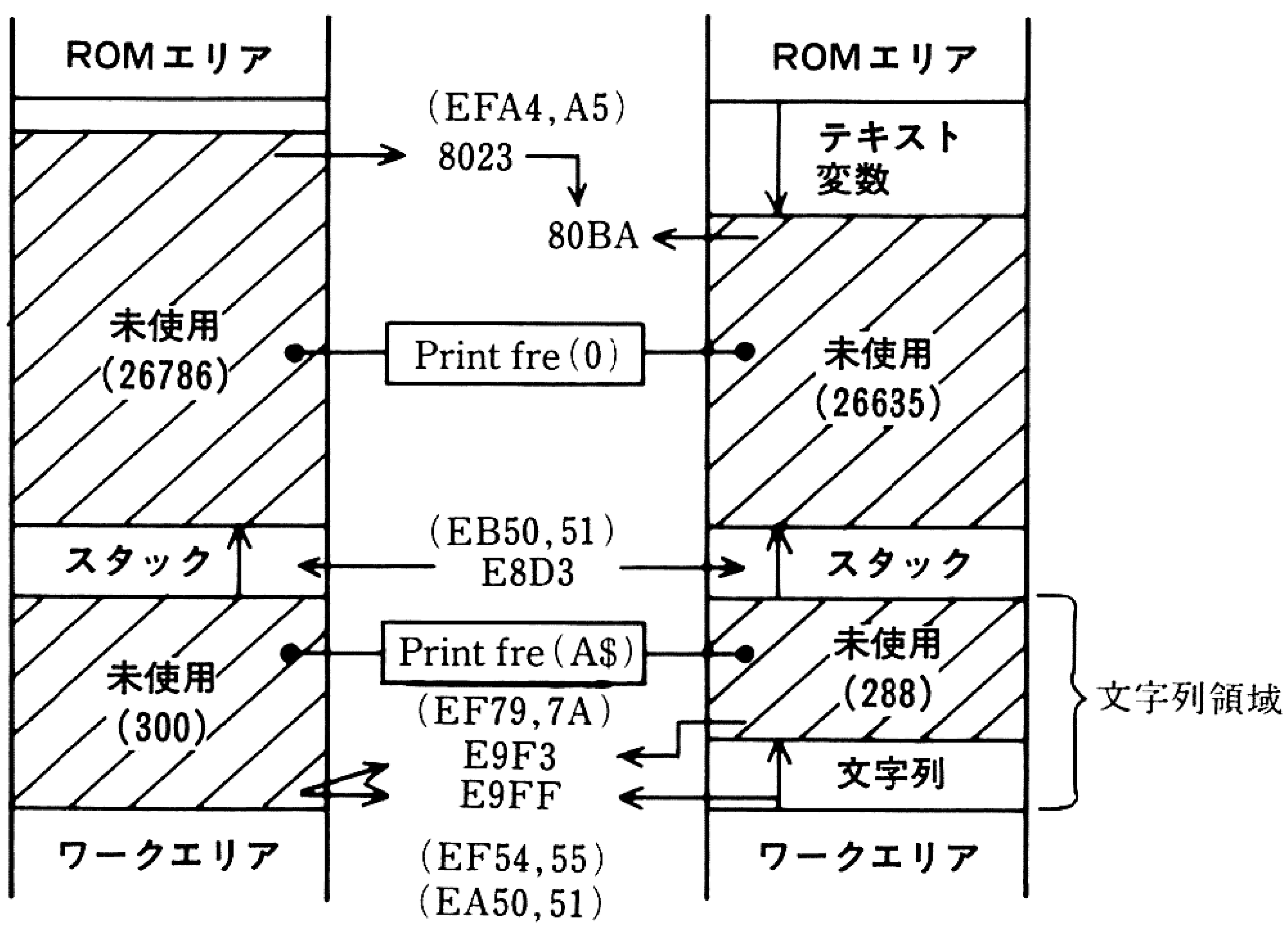
run
Phone No. ? 092-714-6236
Ok

print fre(0),fre(a$)
 26635         288
Ok
```

```
mon
*DEFA4
EFA4 BA 80  64 80 04 04 04 04 04 04 04 04
EFB0 04 04 04 04
*DEB50
EB50 D3 E8  FF FF 21 80 20 B7 00 00 00 28 11 29 00 00
*DEF79
EF79 F3 E9  00 EB 5D 86 3F
EF80 80 78 00 00 00 4A 80 00 00
*DEF54
EF54 FF E9  58 EF 05 60 3B 01 08 CE 02 61
EF60 C8 00 00 00
*DEA50
EA50 FF E9  BD 33 00 FF 00 00 01 00 00 00 50 19 01 FF
*
```

〔図表1-18〕

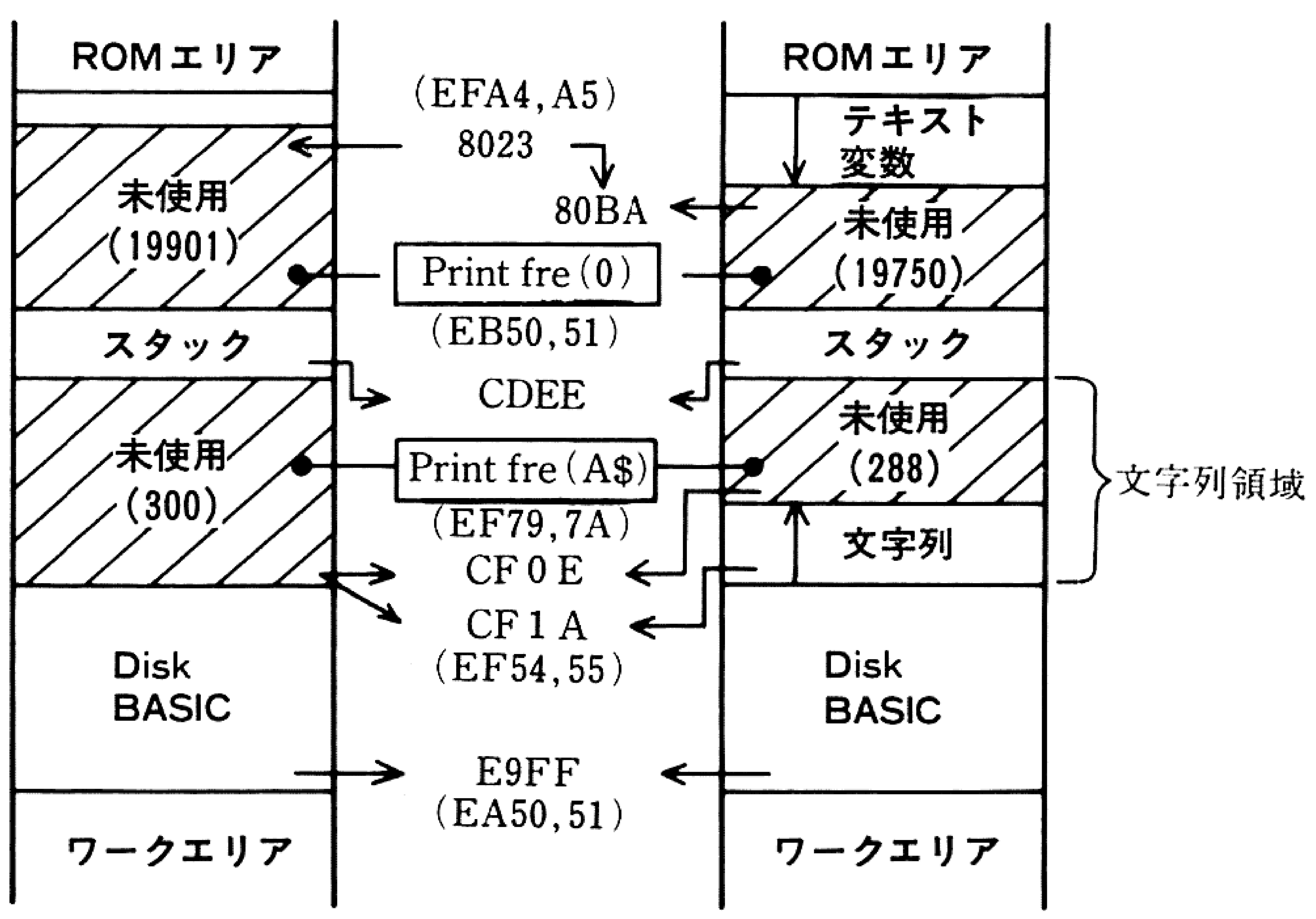
N-BASIC(32K)



〔図表1-19〕

Disk BASIC(32K)

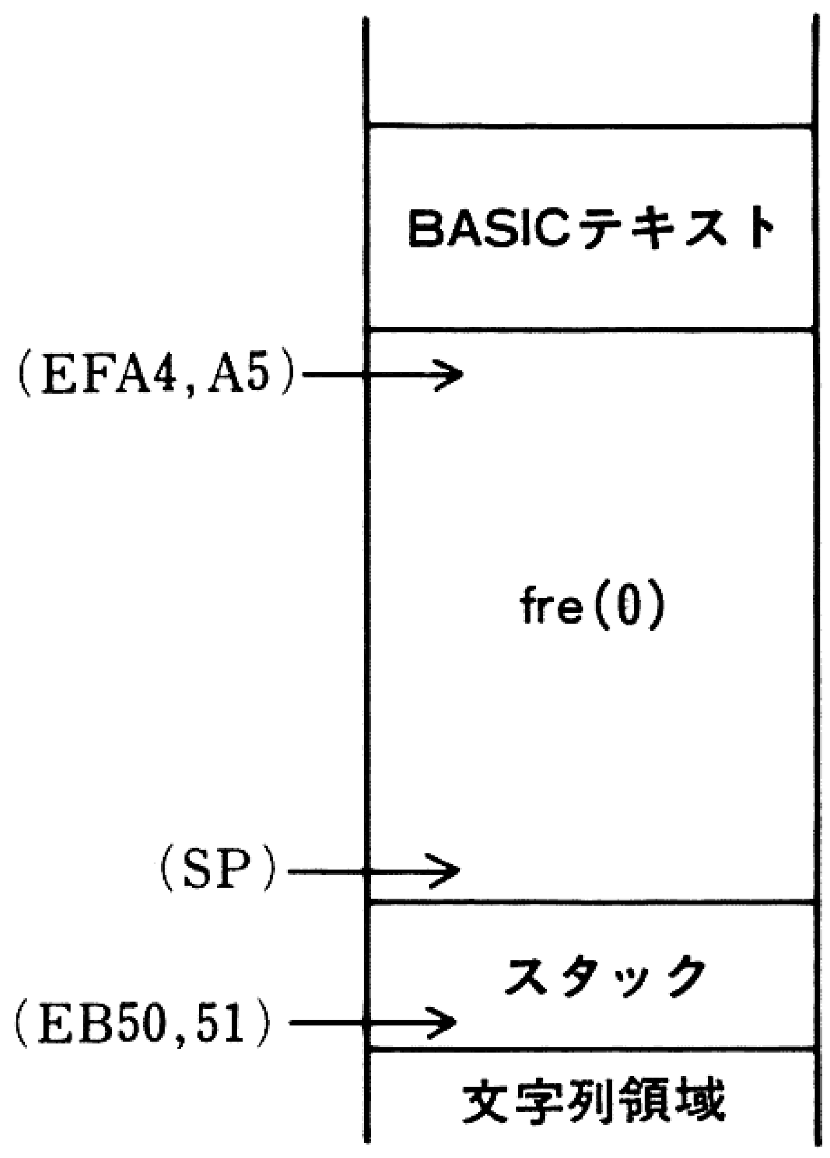
オープンファイルバッファ数により、文字領域およびスタックはアドレスが変わります。



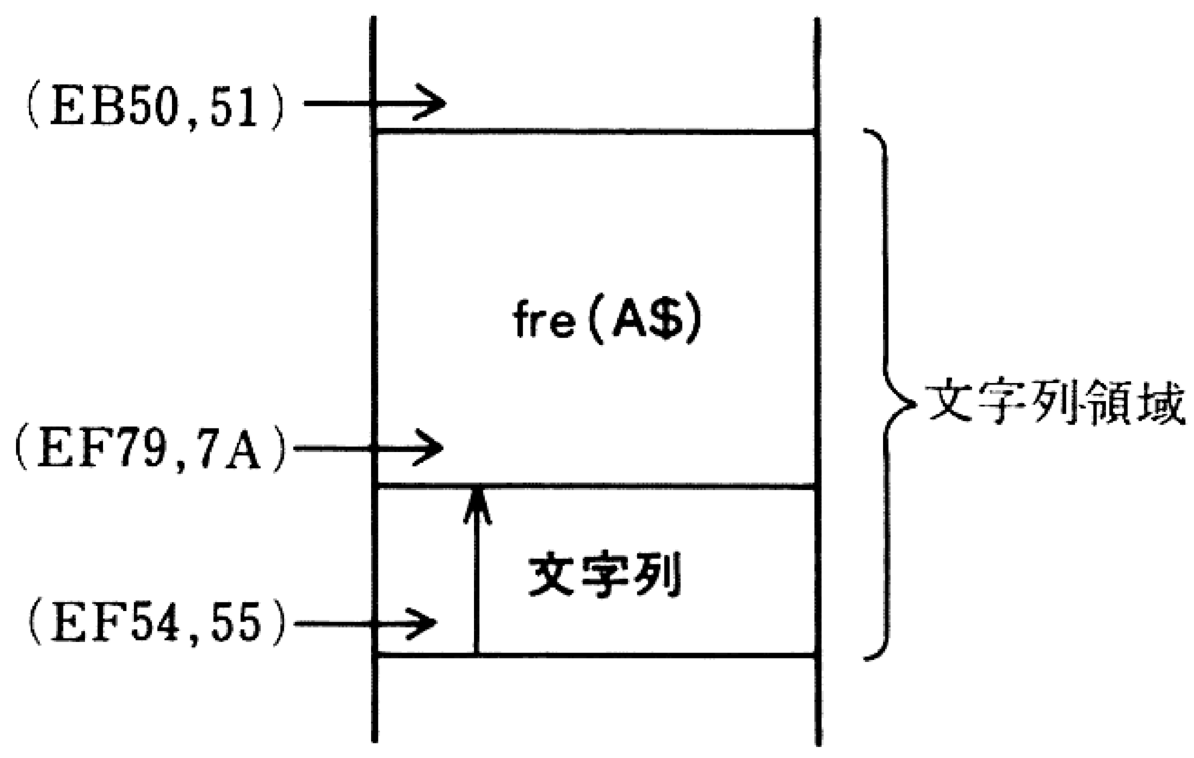
テキスト領域のフリーエリアは、(EFA4,A5)から、Z-80CPU内のSP(スタックポインタ)に格納されているアドレスまでですが、このスタックポインタのアドレスと(EB50,51)に示されるアドレスの間のスタック領域は、機械語のPUSH,POPが実行されるごとに、変化するので、厳密に言えば、fre(0)の値は、処理ルーチンの中でSPのアドレスを読んだ時点でのフリーエリアを示していることになります。

文字列領域は、スタックポインタの始まりアドレス(EB50,51の内容)+1から(EF54,55の内容)までで、文字列は、アドレスの大きい方から、小さい方へと格納されていき、その先頭アドレスが(EF79,7A)に格納されているのでフリーエリア容量は、(EF79,7A) − (EF50,51)=fre(A$)となります。

〔図表1-20〕



〔図表1-21〕



1-12.メモリダンプリスト

メモリの内容を見る場合、16進コードだけでなく、一緒にキャラクタも出力すれば、より内容が解りやすくなります。

次にそのプログラム例を示します。

```
list
100 '      *****  MEMORY DUMP LIST  *****
110 WIDTH 80,25:CONSOLE 0,25,0:DEFINT E-K
120 INPUT "Start Address   ";S$:I=VAL("&h"+S$)
130 INPUT "End   Address   ";E$:E=VAL("&h"+E$):PRINT
140 IF I MOD 16<>0 THEN I=I-1:GOTO140
150 G=0:PRINT RIGHT$("000"+HEX$(I),4)"  ";
160 FOR J=0 TO 15:K=PEEK(I+J):PRINT RIGHT$("0"+HEX$(K),2)" ";
170 IF J=7 OR J=15 THEN PRINT " ";
180 G=G+K:NEXT:PRINT RIGHT$("00"+HEX$(G),3)"  ";
190 FOR J=0 TO 15:K=PEEK(I+J)
200 IF K<32 OR K>247 THEN PRINT "・"; ELSE PRINT CHR$(K);
210 IF J=7 THEN PRINT " ";
220 NEXT:PRINT:I=I+16:IF I<E THEN 150 ELSE 120
OK
```

実行すると、Start AddressとEnd Addressを聞いてきますので、16進コードで希望するアドレス(0000～FFFF)を入力してください。

このプログラムは、CRT上に出力するようにしていますが、プリンタに出力したい時は150行以降のPRINT(8ヶ所)をすべてLPRINTに変更するか、画面プリントの項を参照してください。また、グラフィックの出ないプリンタの場合は、200行のKの条件を変えてください。

次に実行例を示します。中間言語テーブル(34F5～37BB)の一部、およびファンクションキー初期データ(3A80～3B20)の部分を見ています。

```
run
Start Address   ? 34f0
End   Address   ? 35f0                                           チェックサム
                                                                     ↓
34F0  37 A6 37 A7 37 4E C4 F8  42 D3 06 54 CE 0E 53 C3  75D  7ｦ7ｧ7NﾄＸ･ Bﾓ･Tﾎ･Sﾃ
3500  15 55 54 CF A9 54 54 52  A4 E7 00 43 44 A4 1E 45  649  ･UTﾏｩTTR ､◤･CD､･E
3510  45 D0 B2 00 4F 4E 53 4F  4C C5 9F 4C 4F 53 C5 CA  733  Eﾐｲ･ONSO Lﾅ╯LOSﾅﾊ
3520  4F 4E D4 99 4C 45 41 D2  92 4C 4F 41 C4 9B 53 41  70F  ONﾔ┐LEAﾒ ┤LOAﾄ┘SA
3530  56 C5 9A 53 52 4C 49 CE  E6 49 4E D4 1F 53 4E C7  795  Vﾅ└SRLIﾎ ◥INﾔ･SNｷ
3540  20 44 42 CC 21 56 C9 23  56 D3 24 56 C4 25 4F D3  683   DBﾌ!Vﾉ# Vﾓ$Vﾄ%Oﾓ
3550  0C 48 52 A4 16 4D C4 B8  4F 4C 4F D2 B5 00 41 54  62F  ･HR､･Mﾄｸ OLOﾒｵ･AT
3560  C1 84 49 CD 86 45 46 53  54 D2 AB 45 46 49 4E D4  786  ﾁ▄Iﾍ█EFS TﾒｫEFINﾔ
3570  AC 45 46 53 4E C7 AD 45  46 44 42 CC AE 53 4B 4F  6C4  ｬEFSNｷｭE FDBﾌｮSKO
3580  A4 C2 45 C6 97 45 4C 45  54 C5 A8 53 4B 49 A4 E8  812  ､ﾂEﾆ▕ELE TﾅｨSKI､♠
3590  53 4B C6 26 45 C3 1D 41  54 45 A4 EB 00 4E C4 81  6AB  SKﾆ&Eﾃ･A TE､♣･Nﾄ▁
35A0  4C 53 C5 A1 52 41 53 C5  A5 52 52 4F D2 A6 52 CC  7DE  LSﾅ｡RASﾅ ･RROﾒｦRﾌ
35B0  DF 52 D2 E0 58 D0 0B 4F  C6 27 51 D6 FB 00 4F 52  815  ﾟRﾒ═Xﾐ･O ﾆ'Qﾖ･･OR
35C0  4D 41 D4 B3 4F D2 82 49  45 4C C4 C6 49 4C 45 D3  7C9  MAﾔｳOﾒ▂I ELﾄﾆILEﾓ
35D0  CD CE DC 52 C5 0F 49 D8  22 50 4F D3 2A 00 4F 54  71F  ﾍﾎﾜRﾅ･Iﾘ "POﾓ*･OT
35E0  CF 89 4F 20 54 CF 89 4F  53 55 C2 8D 45 D4 C7 00  799  ﾏ▎O Tﾏ▎O SUﾂ█Eﾔｷ･

run
Start Address   ? 3a80
End   Address   ? 3b30

3A80  79 09 00 00 00 00 00 00  00 00 00 00 00 00 00 00  082  y······· ········
3A90  00 61 75 74 6F 20 00 00  00 00 00 00 00 00 00 00  1D9  ･auto ･･ ········
3AA0  00 67 6F 20 74 6F 20 00  00 00 00 00 00 00 00 00  1F9  ･go to ･ ········
3AB0  00 6C 69 73 74 20 00 00  00 00 00 00 00 00 00 00  1DC  ･list ･･ ········
3AC0  00 72 75 6E 0D 00 00 00  00 00 00 00 00 00 00 00  162  ･run････ ········
3AD0  00 74 69 6D 65 24 00 00  00 00 00 00 00 00 00 00  1D3  ･time$･･ ········
3AE0  00 6B 65 79 20 00 00 00  00 00 00 00 00 00 00 00  169  ･key ･･･ ········
3AF0  00 70 72 69 6E 74 20 00  00 00 00 00 00 00 00 00  24D  ･print ･ ········
3B00  00 6C 69 73 74 2E 0D 1E  1E 00 00 00 00 00 00 00  233  ･list.･･ ········
3B10  00 63 6F 6E 74 0D 00 00  00 00 00 00 00 00 00 00  1C1  ･cont･･･ ········
3B20  00 00 2F B7 C9 3C C9 3C  C9 C3 00 00 C3 00 00 00  53F  ･･/ｷﾉ<ﾉ< ﾉﾃ･･ﾃ･･･
```

16バイト毎の合計(Check Sum)も出力しますので、機械語の入力ミスのCheckにも利用できます。

1-13.複数プログラムのロード

プログラム開始アドレスのポインタ(EB54,55)の内容を変えることによって、いくつものプログラムをユーザーズエリア上に、LOADまたは作成することができます。

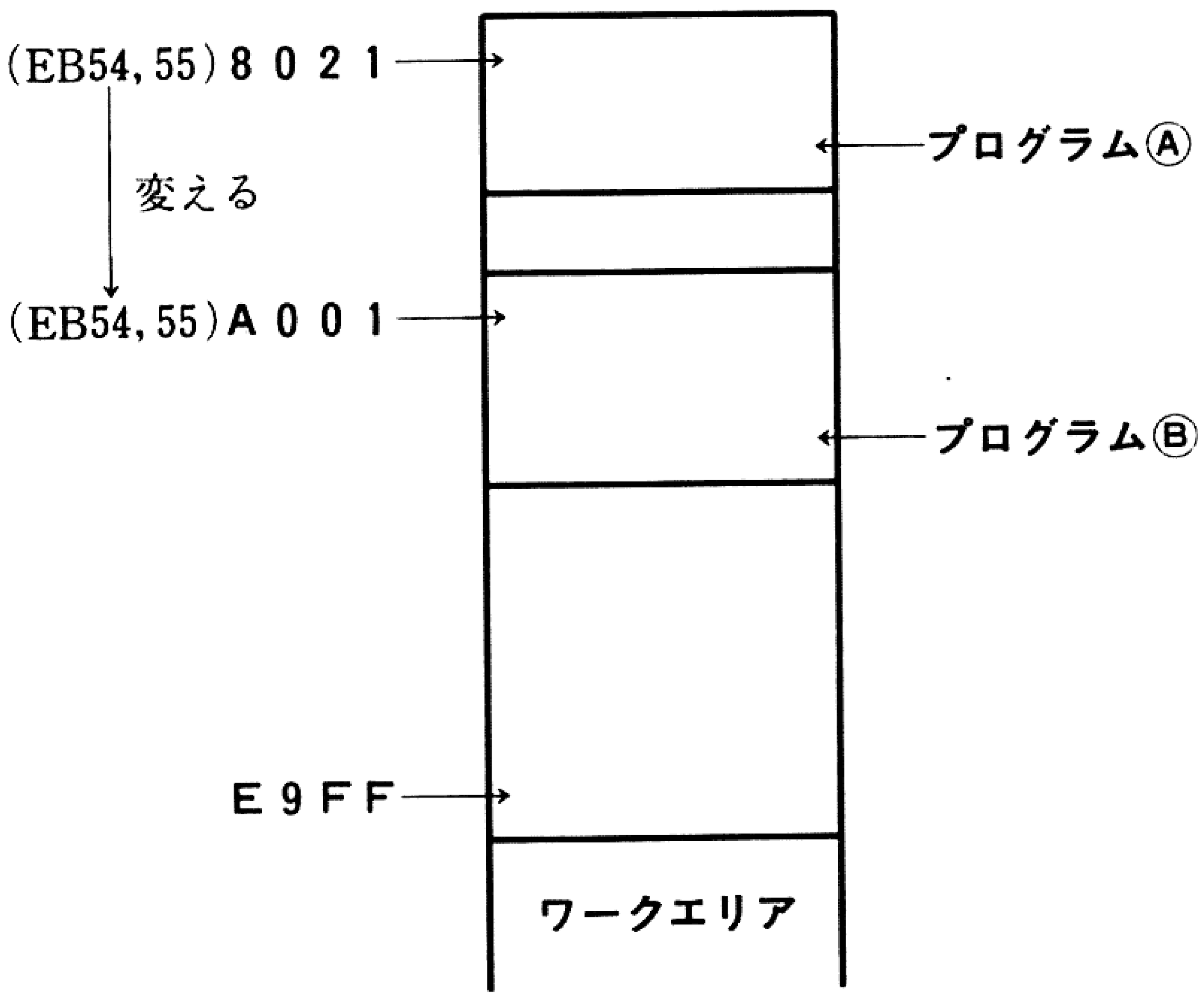
例えば、すでに8020番地からプログラムⒶが入っており、もう一つのプログラムⒷをA000番地よりLOADまたはキーインしようとする場合、

1. A000番地を〝00〟にする。
2. EB54,55番地にA001を入れる。

EB54……01H
EB55……A0H

3. NEWの操作をした後、プログラムをLOAD(CLOAD) あるいはキーインすれば、A001番地からプログラムが入り、LIST 実行が可能となります。

〔図表1-22〕




```
mon
*SA000
A000 FF-00 FF-
*SEB54
EB54 21-01 80-A0 20-
*
```

元に戻す場合は、EB54,55番地に元のプログラムの開始番地8021を入れるだけで、8020から入っているプログラムをlist、実行できます。

実際に試してみましょう。

8020番地から次のプログラムを入れ、実行します。(必ずしも実行する必要はありませんが……)

```
list
100 A$="PC-8001"
110 FOR I=0 TO 2:READ NM$(I):NEXT
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
Ok

run
Phone No. ? 092-714-6236
Ok
```

次に、モニタモードで、ポインタを書き替えた後、前記のDumpプログラムをLOADまたはキーインします。

```
mon
*SA000
A000 FF-00 FF-
*SEB54
EB54 21-01 80-A0 20-
*
Ok
```

```
cload"Dump
Found:Dump
Ok
```

このプログラムを実行すれば8020番地から入れたプログラムのメモリ内の状態をよく観察できます。

```
run
Start Address   ? 8020
End   Address   ? 80af

8020  00 32 80 64 00 41 24 F1  22 50 43 2D 38 30 30 31  417  ·2_d·A$■ "PC-8001
8030  22 00 4B 80 6E 00 82 20  49 F1 11 20 D7 20 13 3A  4AC  "·K_n·▄  I■· ラ ·:
8040  87 20 4E 4D 24 28 49 29  3A 83 00 65 80 78 00 84  49E  █ NM$(I) :▄·e_x·■
8050  20 53 79 73 74 65 6D 2C  53 6F 66 74 2C 46 75 6B  5BF   System, Soft,Fuk
8060  75 6F 6B 61 00 7C 80 82  00 85 20 22 50 68 6F 6E  58A  uoka·¦_▄ ·■ "Phon
8070  65 20 4E 6F 2E 20 22 3B  50 48 24 00 00 00 03 00  2AC  e No. "; PH$·····
8080  41 07 29 80 04 00 49 00  00 40 82 03 48 50 0C 0F  2B6  A·)_··I· ·@▄·HP··
8090  CF 03 4D 4E 24 00 01 0B  00 06 51 80 04 58 80 07  357  マ·MN$··· ··Q_·X_·
80A0  5D 80 00 00 00 00 00 00  00 00 00 00 00 00 00 00  0DD  ]_······ ········
Ok
```

このように、ユーザーエリア内にいくつものプログラムを入れておけば、いちいちLOADする必要がないので非常に便利です。

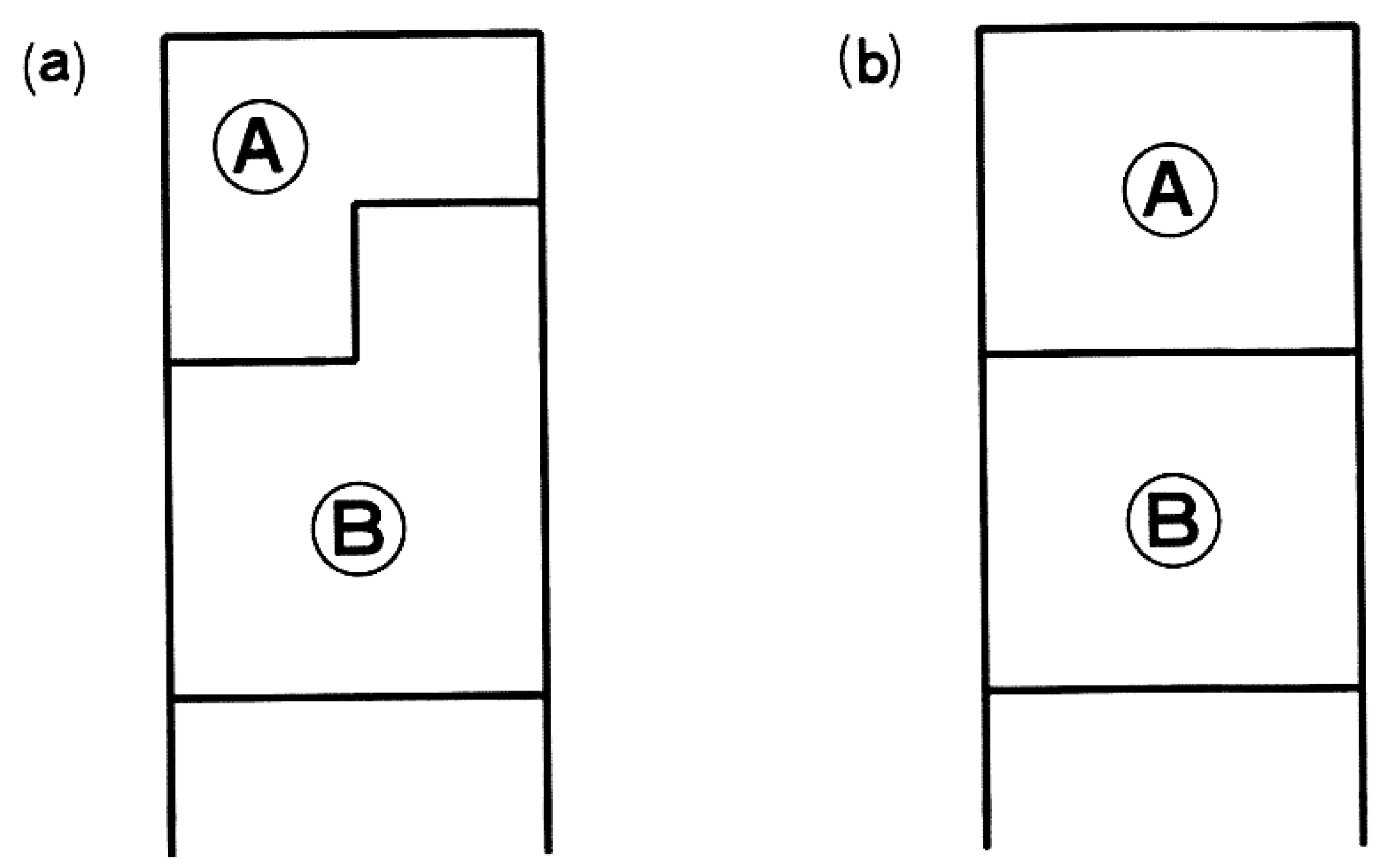
ただし、前の方に入れたプログラムを実行してみて、変数領域がどこまであるかを確認して、その後に入れるプログラムが破壊されないよう十分余裕のあるアドレスから、次のプログラムを入れてください。

1-14.プログラムのアペンド

2つ以上のプログラムをアペンド(結合)する方法について説明します。

Disk BASICシステムの場合には、MERGEというコマンドがあって、本体メモリ内のプログラムⒶと、アスキーSAVEされたディスク内プログラムⒷを結合させることができます。そしてこのコマンドの場合、もしプログラムⒶとプログラムⒷに同じ行番号があったら、Ⓑの方が優先され同じⒶの行番号の内容は、消されてしまいます。また、例えばⒷの110行は、Ⓐの100行と120行の間に入っていき、(a)図のように重ね合わせる形で結合されます。したがってⒷの方の行番号をあらかじめRENUMして、Ⓐの最大行番号より大きくしておけば、(b)図のようにⒷはⒶの後の方に、つけ加わる形で結合されます。

〔図表1-23〕



N-BASICには、このようにプログラムをアペンドするコマンドはありませんが、次の方法で簡単に(b)図のようにプログラムⒶの後にプログラムⒷを結合することができます。複数プログラムのLOADの場合は、テキスト開始番地をプログラムⒶと重ならないよう間をあけて、Ⓑの格納アドレスを設定しましたが、こんどは、Ⓐの直後に設定すればよい訳です。

(1)まずプログラムⒶをキーインするか、テープよりCLOADまたは、ディスクよりLOADします。(LISTして、最大行番号を覚えておきます。)

```
cload"MEMdmp
Ok
```

```
list
100 '      *****  MEMORY DUMP LIST  *****
110 WIDTH 80,25:CONSOLE 0,25,0:DEFINT E-K
120 INPUT "Start Address   ";S$:I=VAL("&h"+S$)
130 INPUT "End   Address   ";E$:E=VAL("&h"+E$):PRINT
 :
 :
200 IF K<32 OR K>247 THEN PRINT "・"; ELSE PRINT CHR$(K);
210 IF J=7 THEN PRINT " ";
220 NEXT:PRINT:I=I+16:IF I<E THEN 150 ELSE 120
Ok
```

(2)プログラムの最後のリンクポインタの先頭アドレスをEB54,55番地(テキスト開始番地が格納されている)に書き込みます。

```
mon
*DEFA0
EFA0 DF 81 DF 81 DF 81 20 80 04 04 04 04 04 04 04 04
*SEB54
EB54 21-DD 80-81 20-
*
Ok
```

〔図表1-24〕　〔図表1-25〕

```
81A0 20 FF 96 28 4B 29 3B 00 [BB 81] D2 00 8B 20 4A F1
81B0 18 20 D8 20 91 20 22 20 22 3B 00 [DD 81] DC 00 83
81C0 3A 91 3A 49 F1 49 F3 0F 10 3A 8B 20 49 F2 45 20
81D0 D8 20 0E 96 00 20 3A A1 20 0E 78 00 00 [00 00] [03
```

プログラムの終わりを示す00, 00のリンクポインタ。

ここのアドレス(81DDH)をEB54,55に書込む

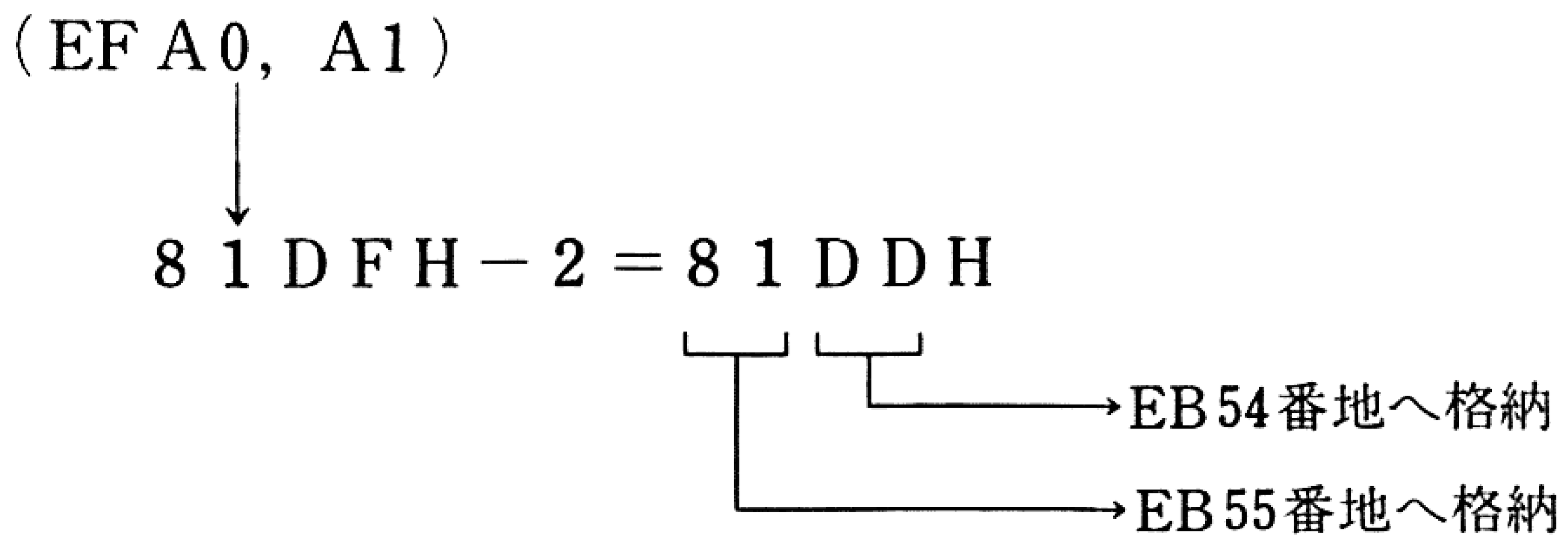
変数領域の始まり

(ここのアドレスは、EFA0, A1に格納されています。)

すなわち、プログラムⒶの最後のリンクポインタのアドレスからプログラムⒷが書込まれるように、テキスト開始番地(EB54,55の内容)を書き換えます。

EB54,55に書込むべきアドレスは、EFA0, A1の内容(単純変数領域の始まりアドレス)から2だけ引いたものです。

〔図表1-26〕



(3)プログラム⒝をキーインするか、テープよりCLOADまたは、DiskよりLOADします。

```
cload"STMcpy
OK
```

ディスクの場合、LOADするプログラムは、アスキー形式、非アスキー形式のどちらでもかまいません。

listして、プログラムⒶと行番号がダブっていないか確認します。

```
list
100 REMOVE:CLEAR 1000:DIM S$(15),C$(15)
110 FIELD#0,70AS F$
120 FOR I=0 TO15:FIELD#0, I*16AS DY$,1AS S$(I),9AS DY$,1AS C$(15):NEXT
 .
 .
 .
410 FOR S=14 TO 16:DSKO$ 2,18,S:NEXT
420 FOR T=0 TO 2:FOR S=1 TO 16:DY$=DSKI$(1,T,S):DSKO$ 2,T,S:NEXT S,T
430 PRINT:PRINT "Complete"
OK
```

この例のようにⒶとダブっている場合は、RENUMします。

```
renum 300
OK
```

(4)EB 54, 55に格納されているテキスト開始番地を初めの8021 Hに戻します。

（16Kシステムの場合はC021 H）

```
mon
*SEB54
EB54 DD-21 81-80 20-
*
OK
```

これでアペンド完了です。つなぎ目の部分をLIST してみましょう。

プログラムⒶ

```
list 210,320
210 IF J=7 THEN PRINT " ";
220 NEXT:PRINT:I=I+16:IF I<E THEN 150 ELSE 120
300 REMOVE:CLEAR 1000:DIM S$(15),C$(15)
310 FIELD#0,70AS F$
320 FOR I=0 TO15:FIELD#0, I*16AS DY$,1AS S$(I),9AS DY$,1AS C$(15):NEXT
OK
```

プログラムⒷ

必要であれば再度RENUM してください。

もし、(3)でRENUMせずに、行番号がダブったままですと、つなぎ目の部分は、

```
list 210,
210 IF J=7 THEN PRINT " ";
220 NEXT:PRINT:I=I+16:IF I<E THEN 150 ELSE 120
100 REMOVE:CLEAR 1000:DIM S$(15),C$(15)
110 FIELD#0,70AS F$
120 FOR I=0 TO15:FIELD#0, I*16AS DY$,1AS S$(I),9AS DY$,1AS C$(15):NEXT
130 PRINT "Drive 1 ... System Disk"
```

220行の後に100行が表示されてしまいます。この状態でRENUMすると、プログラムⒷ内のGOTO, GOSUB, THEN などの後に続く行番号は、元のままで変わらず、正常な実行をしなくなりますのでご注意ください。

こうしてつくられたプログラムの後に、プログラムⒸをアペンドすることも同様の方法で容易にできます。

第2章　CRTディスプレイ

第2章　CRTディスプレイ

2-1. WIDTHとVRAMの関係

WIDTH文により、画面に表示する文字の行数と桁数を指定しますが、N-BASICで指定できる組合わせは、次の8通りです。

〔図表2-1〕

WIDTH

〈桁　数〉	〈行　数〉
36	20
40	20
72	25
80	25

起動時は、40桁20行になっています。

また、画面に表示される内容は、VRAM（ビデオラム）と呼ばれる、F300～FEB7番地に納められているものであり、そのVRAMの領域をDMA（Direct Memory Access）コントローラ（μPD8257）の制御により、CPUを一時停止させながら直接データを読み取り、CRTへ表示します。

画面の1文字がVRAMの1バイトに対応しています。（白黒モードのとき左上隅がF300番地の内容）

したがって、画面1行分（80文字）が、VRAM80バイトに対応しており（40文字モードのときは、1バイト置き）、続く40バイト（20組）が、その行の属性を定義する、アトリビュート（属性）領域となっています。したがって、VRAM領域は、120バイト×25行＝3Kバイトあり、うち、2Kバイトが文字コード領域で1Kバイトがアトリビュート領域となります。

画面とVRAMアドレス対応表

2-1-1 40文字モードアドレスマップ

〔図表2-2〕

40桁／白黒モード（0〜39桁）
40桁／カラーモード(39桁)（1〜39桁）
36桁モード（2〜37桁）
アトリビュートエリア

20行（0〜19行）　25行（0〜24行）

行	0	1	2	3	4	5		36	37	38	39	アトリビュートエリア						
0	F300	F302	F304	F306	F308	F30A	・・・・・・・・・・・・・・	F348	F34A	F34C	F34E	F350	F351	F352	F353	・・・・・	F376	F377
1	F378	F37A	F37C	F37E	F380	F382	・・・・・・・・・・・・・・	F3C0	F3C2	F3C4	F3C6	F3C8	F3C9	F3CA	F3CB	・・・・・	F3EE	F3EF
2	F3F0	F3F2	F3F4	F3F6	F3F8	F3FA	・・・・・・・・・・・・・・	F438	F43A	F43C	F43E	F440	F441	F442	F443	・・・・・	F466	F467
3	F468	F46A	F46C	F46E	F470	F472	・・・・・・・・・・・・・・	F4B0	F4B2	F4B4	F4B6	F4B8	F4B9	F4BA	F4BB	・・・・・	F4DE	F4DF
4	F4E0	F4E2	F4E4	F4E6	F4E8	F4EA	・・・・・・・・・・・・・・	F528	F52A	F52C	F52E	F530	F531	F532	F533	・・・・・	F556	F557
5	F558	F55A	F55C	F55E	F560	F562	・・・・・・・・・・・・・・	F5A0	F5A2	F5A4	F5A6	F5A8	F5A9	F5AA	F5AB	・・・・・	F5CE	F5CF
6	F5D0	F5D2	F5D4	F5D6	F5D8	F5DA	・・・・・・・・・・・・・・	F618	F61A	F61C	F61E	F620	F621	F622	F623	・・・・・	F646	F647
7	F648	F64A	F64C	F64E	F650	F652	・・・・・・・・・・・・・・	F690	F692	F694	F696	F698	F699	F69A	F69B	・・・・・	F6BE	F6BF
8	F6C0	F6C2	F6C4	F6C6	F6C8	F6CA	・・・・・・・・・・・・・・	F708	F70A	F70C	F70E	F710	F711	F712	F713	・・・・・	F736	F737
9	F738	F73A	F73C	F73E	F740	F742	・・・・・・・・・・・・・・	F780	F782	F784	F786	F788	F789	F78A	F78B	・・・・・	F7AE	F7AF
10	F7B0	F7B2	F7B4	F7B6	F7B8	F7BA	・・・・・・・・・・・・・・	F7F8	F7FA	F7FC	F7FE	F800	F801	F802	F803	・・・・・	F826	F827
11	F828	F82A	F82C	F82E	F830	F832	・・・・・・・・・・・・・・	F870	F872	F874	F876	F878	F879	F87A	F87B	・・・・・	F89E	F89F
12	F8A0	F8A2	F8A4	F8A6	F8A8	F8AA	・・・・・・・・・・・・・・	F8E8	F8EA	F8EC	F8EE	F8F0	F8F1	F8F2	F8F3	・・・・・	F916	F917
13	F918	F91A	F91C	F91E	F920	F922	・・・・・・・・・・・・・・	F960	F962	F964	F966	F968	F969	F96A	F96B	・・・・・	F98E	F98F
14	F990	F992	F994	F996	F998	F99A	・・・・・・・・・・・・・・	F9D8	F9DA	F9DC	F9DE	F9E0	F9E1	F9E2	F9E3	・・・・・	FA06	FA07
15	FA08	FA0A	FA0C	FA0E	FA10	FA12	・・・・・・・・・・・・・・	FA50	FA52	FA54	FA56	FA58	FA59	FA5A	FA5B	・・・・・	FA7E	FA7F
16	FA80	FA82	FA84	FA86	FA88	FA8A	・・・・・・・・・・・・・・	FAC8	FACA	FACC	FACE	FAD0	FAD1	FAD2	FAD3	・・・・・	FAF6	FAF7
17	FAF8	FAFA	FAFC	FAFE	FB00	FB02	・・・・・・・・・・・・・・	FB40	FB42	FB44	FB46	FB48	FB49	FB4A	FB4B	・・・・・	FB6E	FB6F
18	FB70	FB72	FB74	FB76	FB78	FB7A	・・・・・・・・・・・・・・	FBB8	FBBA	FBBC	FBBE	FBC0	FBC1	FBC2	FBC3	・・・・・	FBE6	FBE7
19	FBE8	FBEA	FBEC	FBEE	FBF0	FBF2	・・・・・・・・・・・・・・	FC30	FC32	FC34	FC36	FC38	FC39	FC3A	FC3B	・・・・・	FC5E	FC5F
20	FC60	FC62	FC64	FC66	FC68	FC6A	・・・・・・・・・・・・・・	FCA8	FCAA	FCAC	FCAE	FCB0	FCB1	FCB2	FCB3	・・・・・	FCD6	FCD7
21	FCD8	FCDA	FCDC	FCDE	FCE0	FCE2	・・・・・・・・・・・・・・	FD20	FD22	FD24	FD26	FD28	FD29	FD2A	FD2B	・・・・・	FD4E	FD4F
22	FD50	FD52	FD54	FD56	FD58	FD5A	・・・・・・・・・・・・・・	FD98	FD9A	FD9C	FD9E	FDA0	FDA1	FDA2	FDA3	・・・・・	FDC6	FDC7
23	FDC8	FDCA	FDCC	FDCE	FDD0	FDD2	・・・・・・・・・・・・・・	FE10	FE12	FE14	FE16	FE18	FE19	FE1A	FE1B	・・・・・	FE3E	FE3F
24	FE40	FE42	FE44	FE46	FE48	FE4A	・・・・・・・・・・・・・・	FE88	FE8A	FE8C	FE8E	FE90	FE91	FE92	FE93	・・・・・	FEB6	FEB7

locate x,yの位置のVRAMアドレスは、

40桁／白黒モード　AD＝&HF300＋120＊y＋2＊x

40桁／カラーモード　AD＝&HF302＋120＊y＋2＊x

（左側1桁が表示されず39桁となります。）

36桁モード　AD＝&HF304＋120＊y＋2＊x

（左右各々2桁が表示されず、00が格納されます。）

40文字モードでは、メモリが1つおきに使用されます。

2-1-2 80文字モードアドレスマップ

〔図表2-3〕

80桁／白黒モード（桁0〜79）
80桁／カラーモード(79桁)（桁1〜79）
72桁モード（桁4〜75）
アトリビュートエリア

20行（行0〜19）　25行（行0〜24）

行	0	1	2	3	4	5		76	77	78	79							
0	F300	F301	F302	F303	F304	F305	…………	F34C	F34D	F34E	F34F	F350	F351	F352	F353	…..	F376	F377
1	F378	F379	F37A	F37B	F37C	F37D	…………	F3C4	F3C5	F3C6	F3C7	F3C8	F3C9	F3CA	F3CB	…..	F3EE	F3EF
2	F3F0	F3F1	F3F2	F3F3	F3F4	F3F5	…………	F43C	F43D	F43E	F43F	F440	F441	F442	F443	…..	F466	F467
3	F468	F469	F46A	F46B	F46C	F46D	…………	F4B4	F4B5	F4B6	F4B7	F4B8	F4B9	F4BA	F4BB	…..	F4DE	F4DF
4	F4E0	F4E1	F4E2	F4E3	F4E4	F4E5	…………	F52C	F52D	F52E	F52F	F530	F531	F532	F533	…..	F556	F557
5	F558	F559	F55A	F55B	F55C	F55D	…………	F5A4	F5A5	F5A6	F5A7	F5A8	F5A9	F5AA	F5AB	…..	F5CE	F5CF
6	F5D0	F5D1	F5D2	F5D3	F5D4	F5D5	…………	F61C	F61D	F61E	F61F	F620	F621	F622	F623	…..	F646	F647
7	F648	F649	F64A	F64B	F64C	F64D	…………	F694	F695	F696	F697	F698	F699	F69A	F69B	…..	F6BE	F6BF
8	F6C0	F6C1	F6C2	F6C3	F6C4	F6C5	…………	F70C	F70D	F70E	F70F	F710	F711	F712	F713	…..	F736	F737
9	F738	F739	F73A	F73B	F73C	F73D	…………	F784	F785	F786	F787	F788	F789	F78A	F78B	…..	F7AE	F7AF
10	F7B0	F7B1	F7B2	F7B3	F7B4	F7B5	…………	F7FC	F7FD	F7FE	F7FF	F800	F801	F802	F803	…..	F826	F827
11	F828	F829	F82A	F82B	F82C	F82D	…………	F874	F875	F876	F877	F878	F879	F87A	F87B	…..	F89E	F89F
12	F8A0	F8A1	F8A2	F8A3	F8A4	F8A5	…………	F8EC	F8ED	F8EE	F8EF	F8F0	F8F1	F8F2	F8F3	…..	F916	F917
13	F918	F919	F91A	F91B	F91C	F91D	…………	F964	F965	F966	F967	F968	F969	F96A	F96B	…..	F98E	F98F
14	F990	F991	F992	F993	F994	F995	…………	F9DC	F9DD	F9DE	F9DF	F9E0	F9E1	F9E2	F9E3	…..	FA06	FA07
15	FA08	FA09	FA0A	FA0B	FA0C	FA0D	…………	FA54	FA55	FA56	FA57	FA58	FA59	FA5A	FA5B	…..	FA7E	FA7F
16	FA80	FA81	FA82	FA83	FA84	FA85	…………	FACC	FACD	FACE	FACF	FAD0	FAD1	FAD2	FAD3	…..	FAF6	FAF7
17	FAF8	FAF9	FAFA	FAFB	FAFC	FAFD	…………	FB44	FB45	FB46	FB47	FB48	FB49	FB4A	FB4B	…..	FB6E	FB6F
18	FB70	FB71	FB72	FB73	FB74	FB75	…………	FBBC	FBBD	FBBE	FBBF	FBC0	FBC1	FBC2	FBC3	…..	FBE6	FBE7
19	FBE8	FBE9	FBEA	FBEB	FBEC	FBED	…………	FC34	FC35	FC36	FC37	FC38	FC39	FC3A	FC3B	…..	FC5E	FC5F
20	FC60	FC61	FC62	FC63	FC64	FC65	…………	FCAC	FCAD	FCAE	FCAF	FCB0	FCB1	FCB2	FCB3	…..	FCD6	FCD7
21	FCD8	FCD9	FCDA	FCDB	FCDC	FCDD	…………	FD24	FD25	FD26	FD27	FD28	FD29	FD2A	FD2B	…..	FD4E	FD4F
22	FD50	FD51	FD52	FD53	FD54	FD55	…………	FD9C	FD9D	FD9E	FD9F	FDA0	FDA1	FDA2	FDA3	…..	FDC6	FDC7
23	FDC8	FDC9	FDCA	FDCB	FDCC	FDCD	…………	FE14	FE15	FE16	FE17	FE18	FE19	FE1A	FE1B	…..	FE3E	FE3F
24	FE40	FE41	FE42	FE43	FE44	FE45	…………	FE8C	FE8D	FE8E	FE8F	FE90	FE91	FE92	FE93	…..	FEB6	FEB7

locate x,yの位置のVRAMアドレスは、

80桁／白黒モード　AD＝&HF300＋120＊y＋x

80桁／カラーモード　AD＝&HF301＋120＊y＋x

（左側1桁が表示されません。）

72桁モード　AD＝&HF304＋120＊y＋x

（左右各々4桁が表示されず00が格納されます。）

VRAMのアドレスに直接poke文でアスキーコードを書込めば、画面に、そのアスキー文字が表示されます。

例えば40文字／白黒モードのとき、

AD＝&HF300＋120＊5＋2＊10

poke AD, &H48

とすると、第5行、10桁目に〝H〞が表示されます。

前記は、locate 10,5：print"H" を意味しますが、カーソルは、全く移動しません。

また、このpoke文によりVRAMに直接書込む方法によれば、00H～1FHのコントロールキャラクタをも画面に表示することができます。

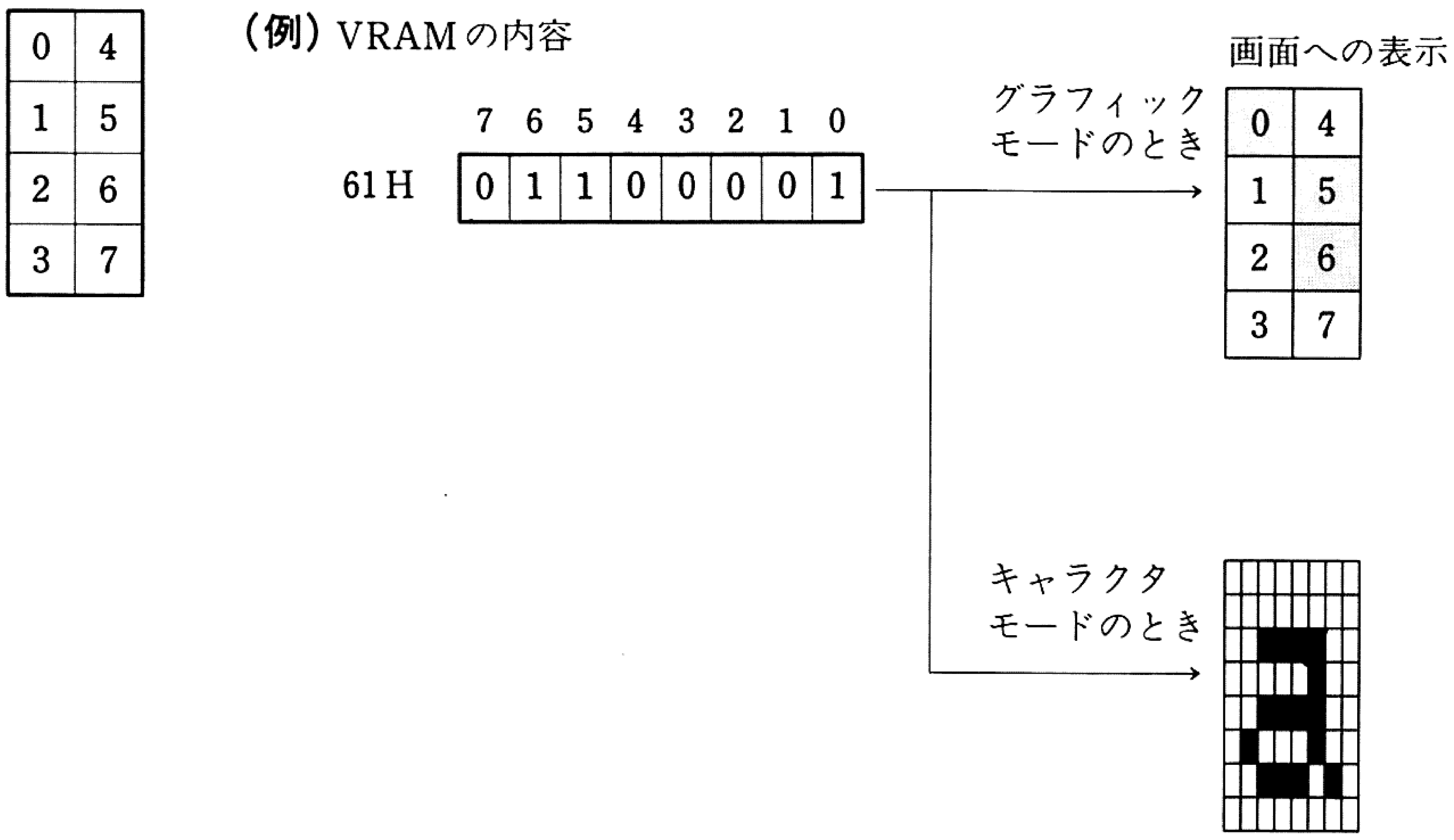
例. poke &HF320,3 …………"E_X"

poke &HF320, &H1C…"→"

ついでですが、36桁モード、72桁モード時、使用していない、左右2～4桁の部分にもpoke文を使って、対応するVRAMにアスキーコードを直接書込めば、そのキャラクタが画面に表示されます。

属性がグラフィックモードのときは、キャラクタ1文字分のエリアが8ドットに分割され下図のように割当てられています。

〔図表2-4〕



2-2.アトリビュートエリアとは

アトリビュートエリアに格納されているコードにより、VRAM文字コード領域のデータをどういう形式(カラー／白黒、グラフィック／キャラクタ、ノーマル／リバース／ブリンク／シークレット、アンダーライン／アッパーライン）で表示するかを定義します。

その格納状態と意味は、次の図のようになります。

〔図表2-5〕

アトリビュートエリアの先頭
1つの属性は、2バイトで表現されている。

F350	F351	F352	F353		F376	F377
桁数	属性	桁数	属性		桁数	属性

属性コード

この桁から次の1バイトの属性コードに従った表示を行なう。

1行につき40バイト(20組)確保されている。
従って、属性の変化は、20回まで許される。

✳属性コード

〔図表2-6〕
(a)白黒モード (CONSOLE ,,, 0)

7	6	5	4	3	2	1	0
Graphic / Character	0	アンダーライン	アッパーライン	0	リバース	ブリンク	シークレット

0 0 0	ノーマル	COLOR 0
0 0 1	シークレット	COLOR 1
0 1 0	ブリンク	COLOR 2
0 1 1	シークレット	COLOR 3
1 0 0	リバース	COLOR 4
1 0 1	リバースシークレット	COLOR 5
1 1 0	リバースブリンク	COLOR 6
1 1 1	リバースシークレット	COLOR 7

(b)カラーモード (CONSOLE ,,, 1)

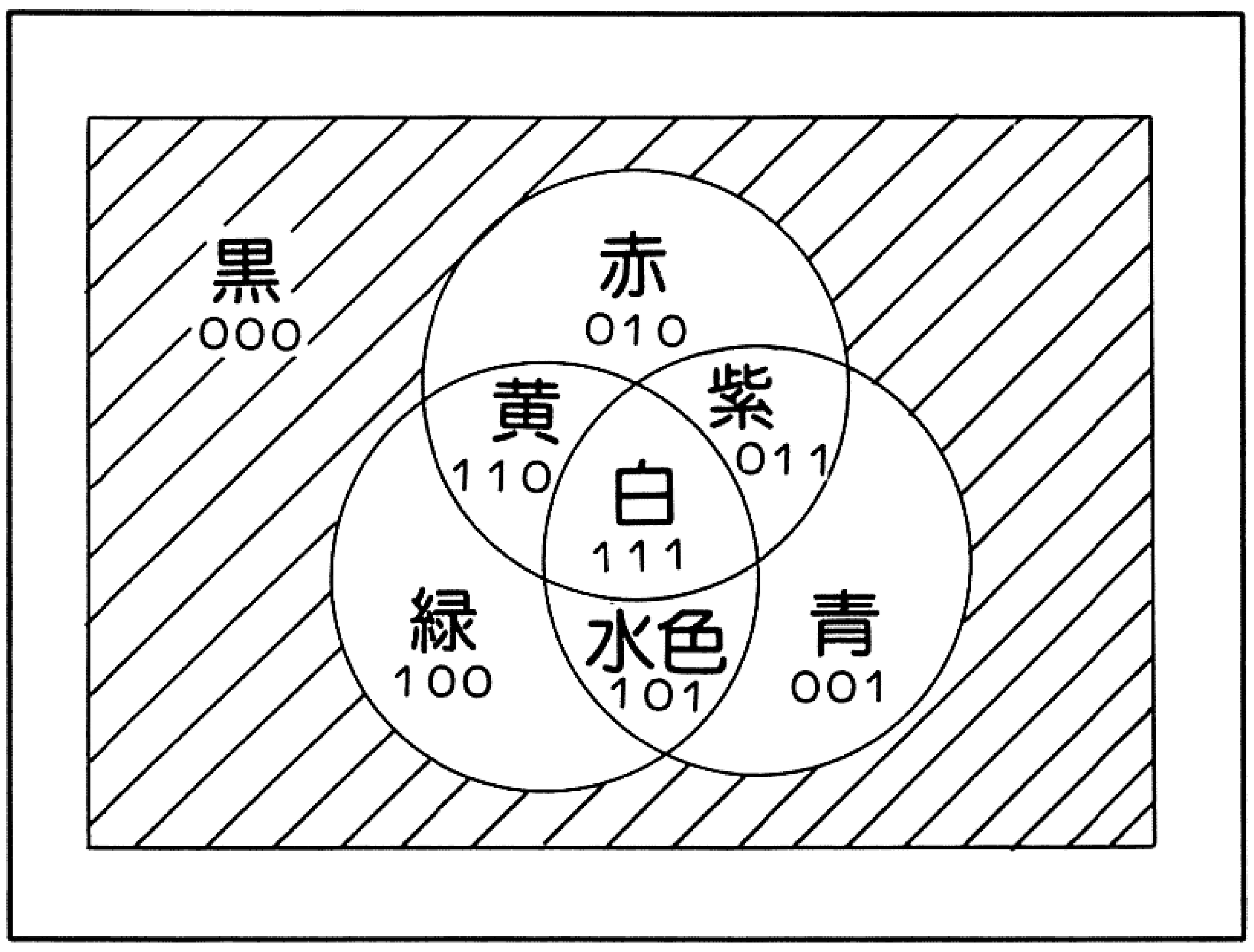
7	6	5	4	3	2	1	0
0	0	アンダーライン	アッパーライン	0 ✳	リバース	ブリンク	シークレット

✳カラー時のLINE文に対応

このビットが0か1かで2通りある。

7	6	5	4	3	2	1	0
緑	赤	青	Graphic / Character	1	0	0	0

0 0 0	黒	COLOR 0	1 0 0	緑	COLOR 4
0 0 1	青	1	1 0 1	水色(シャン)	5
0 1 0	赤	2	1 1 0	黄	6
0 1 1	紫(マゼンタ)	3	1 1 1	白	7

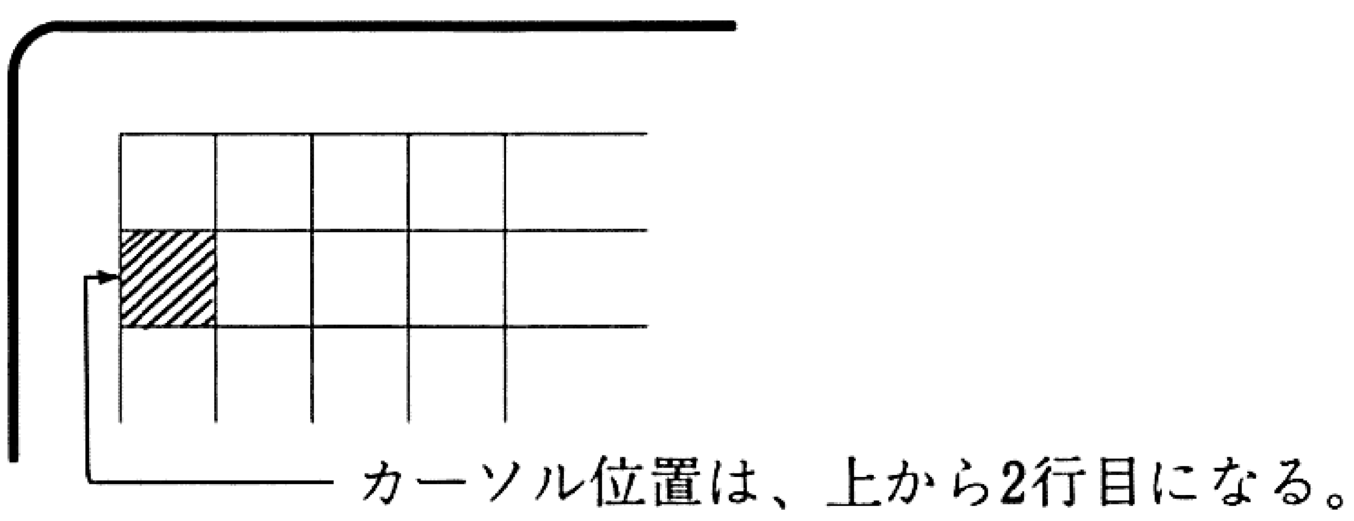


2-3.PRINT文のちょっとした違い

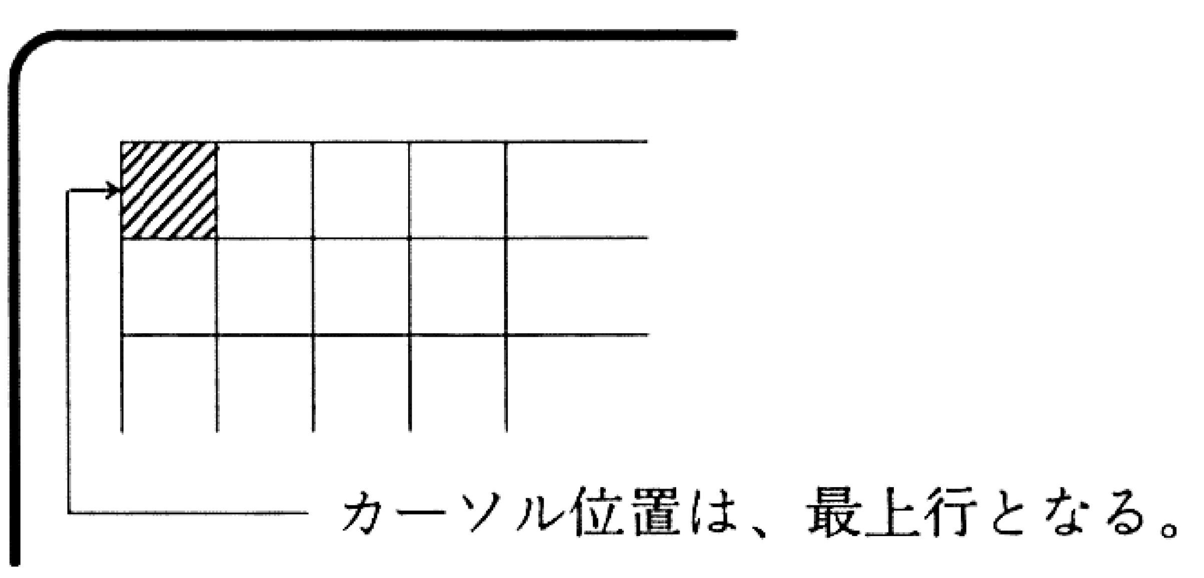
2-3-1 PRINT CHR$(12)とPRINT CHR$(12);

画面をクリアするコントロールコードに0CH(10進数で12)があり、PRINT CHR$(12)で画面をクリアできますが、CHR$(12)の後に、〔;〕があるかないかで、画面クリア後のカーソルの位置が異なります。

PRINT CHR$(12)



PRINT CHR$(12);



ただし、PRINT CHR$(12)に続けて、文字や文字列、その他をPRINTする場合は、PRINT CHR$(12)"ABCD"……と〔;〕を省略しても"ABCD"は、最上行に表示されます。

2-3-2 PRINT(USING)文のいろいろ

(1) PRINTの代りに〔?〕を使うことができます。ダイレクトモード時だけでなく、プログラム作成時も使うことができ、LISTすると、自動的にPRINTに置き換わります。

```
10 ? "Personal Computer"


list
10 PRINT "Personal Computer"
Ok
```

(2) 2つ以上の式を14桁おきに表示するときは〔,〕、続けて表示するときは、〔;〕をその間に入れます。

```
a$="ABC":b$="DEF"
Ok

print a$,b$
ABC           DEF
|←     14桁     →|
Ok
print a$;b$
ABCDEF
Ok
```

(3) 文字定数や文字変数などを続けて表示するとき、間の〔;〕は省略できます。

```
a$="ABC":b$="DEF":c$="GHI"
Ok


print a$b$tab(20)chr$(34)c$chr$(34)
ABCDEF                  "GHI"
Ok
```

```
              ┌─ 省略可
            ┌─┴┐
            ↓  ↓
PRINT A$;B$;TAB(20)………
```

(4) 文字列を表示するとき、後の〔"〕(クオート)は省略できます。

```
print "Address    ←── 後のクオートは省略可
Address
Ok
```

後に、マルチステートメントが続くときは、省略できません。

また、CLOAD,SAVEなどに続くファイル名につける〔"〕も後の分は、省略できます。

```
Cload "Test    ←—— 後のクォートは省略可
Found: Test
```

(5) 数値または、数値変数を表示すると、数値の後に1つスペースが表示されます。その数値が正の数の場合は、前にもスペースが1つ表示され、負の数の場合には、そこに"ー"が表示されます。

```
a=123:b=-456
Ok

print "¥"a"¥"
¥ 123 ¥
Ok
print "¥"b"¥"
¥-456 ¥
Ok

print len(str$(a))
 4
Ok
```

正の数 ␣123␣

負の数 −456␣
↑
負号

したがって(　)の中に数字を詰めて表示したいとき、例えば、Aが正の整数であるとき、

```
print "("a")"
( 123 )
Ok


print "("mid$(str$(a),2)")"
(123)
Ok
```

などとする必要があります。

(6) PRINT USING文のフォーマット文の中に、フォーマット文字(#,&など)以外の文字を書くと、その文字が一緒に表示されます。

```
a=123:b=456
Ok
printusing"### ﾄ ### ｦ ｸﾜｴﾙﾄ #### ﾆ ﾅﾘﾏｽ";a,b,a+b
123 ﾄ 456 ｦ ｸﾜｴﾙﾄ  579 ﾆ ﾅﾘﾏｽ
Ok
```

(7) フォーマット文の後に置く式が、フォーマットの数より多いときは、くりかえしてそのフォーマットが使われます。

```
a=133.45:b=67.8:c=90:d=987.65:e=43.21
Ok
printusing"###.##  ";a,b,c,d,e
133.45   67.80   90.00  987.65   43.21
Ok
```

```
$="ABC":b$="CD":c$="EFG"
Ok
printusing"& & ###.###  ";a$,a,b$,b,c$,c
ABC 133.450  CD   67.800  EFG  90.000
Ok
```

(8) USING文のフォーマット文を文字変数とすることができます。

```
f$="#####.## 円":x$="ｷﾝｶﾞｸ":c=1234.5
Ok
printusing x$+"   "+f$;c
ｷﾝｶﾞｸ   1234.50 円
Ok
```

(9) PRINT文と、TAB関数とUSING文は、続けて使えます。ただし、USING文は、最後にしか使えません。

```
f$="#####.## 円":x$="ｷﾝｶﾞｸ":c=1234.5
Ok
print x$tab(10)"....."using f$;c
ｷﾝｶﾞｸ      ..... 1234.50 円
Ok
```

〔図表2-7〕PRINT USING文一覧表

フォーマット	機　　能	例
!	文字列の最初の文字だけを表示	$="ABCDE" printusing"!";a$ A
&␣␣…␣& n	文字列のうち始めからn文字を左づめで表示	a$="ABCDE" printusing"&　&";a$ ABCD
＃＃＃＃	数値を右づめで表示	a=123 printusing"####";a 　123
＃＃＃.＃＃ ↑ 小数点	指定の場所に小数点を表示	a=12.3 printusing"###.##";a 　12.30 Ok
＋＃＃＃.＃＃ ↑ ＋符号	数値の前に符号(＋または－)を表示	a=12.3:b=-45.6 printusing"+###.##";a,b 　+12.30　-45.60
＃＃＃.＃＃＋ ↑ ＋符号	数値の後に符号(＋または－)を表示	a=12.3:b=-45.6 printusing"###.##+";a,b 　12.30+　45.60-
＃＃＃.＃＃－	負の数のみ後にマイナス記号(－)を表示する。	a=12.3:b=-45.6 printusing"###.##-";a,b 　12.30　45.60-
＊＊＃＃＃.＃ ↑ 2つのアスタリスク	数値領域の頭の空白部分を"＊"で埋める。 (＊＊も桁数に含める。)	a=12.3:b=-45.6 printusing"**###.##";a,b ***12.30　**-45.60
¥¥＃＃＃.＃ ↑ 2つの¥記号	数値の直前に¥を表示する。 (¥¥のうち1個のみ桁数に含める。) (負の数だと－¥…となり使用不可) (指数型にも使用不可)	a=12.3:b=-45.6 printusing"¥¥###.##";a,b 　¥12.30　-¥45.60
＊＊¥＃＃＃.＃ ↑ 2つのアスタリスクと¥記号	数値領域の頭の空白部分を"＊"で埋め数値の直前に¥を表示する。 (＊＊¥のうち2個分桁数に含める)	a=12.3:b=-45.6 printusing"**¥###.## ";a,b ***¥12.30 **-¥45.60
＃＃＃＃＃， コンマ ↑↓ ＃＃＃＃＃，.＃＃	小数点の左側の数値の3桁ごとに〔,〕を表示する。	a=1234.56 printusing"#####,.##";a 　1,234.56
＃＃＃.＃＃^^^^ ↑ 4つの"^"記号	指数形式の表示を指定する。	a=1234.56 printusing"###.##^^^^";a 　12.35E+02

注1）%…指定したフォーマットを越える桁数のときは、前に%が表示される。
注2）桁数の指定が24を越えると、Illegal Function Call エラーになる。

2-3-3 ¥とMODの効果的な使い方

¥とMODの機能は、

¥……整数割り算の商

MOD…整数割り算のあまり

具体的な例で示すと、

14÷3＝4あまり2

↑　　↑

14¥3　14mod 3

ということになります。

小数点以下の部分は、すべて切捨てたあとに計算されます。

```
a=14.99:b=3.1
Ok
?a¥b,a mod b
 4                 2
Ok
```

また、¥、MODで使える数値は、−32768～32767の範囲で、それ以外では、Over flowエラーが生じます。

使い方の例を次に示します。

(1) ある10進数を16進数に換算したときの上位8ビット、下位8ビットに各々分解するとき、

```
a=1234
Ok
print hex$(a¥256),hex$(a mod 256)
4                D2
Ok
```

とすれば、簡単になりますが、もし¥、MODを使わないと、

```
a=1234
Ok
print hex$(int(a/256)),hex$(1234-int(a/256)*256)
4                D2
Ok
```

などとする必要があり、少しめんどうになります。

(2) 数値や文字列を画面に表示するとき、
例えば、1～50の数値を画面に次の様に、12桁毎に、5列、10行で表示したいとき、

1	11	21	31	41
2	12	22	32	42
3	13	23	33	43
4	14	24	34	44
5	15	25	35	45
6	16	26	36	46
7	17	27	37	47
8	18	28	38	48
9	19	29	39	49
10	20	30	40	50

プログラムは、

```
100 ' ■■ ﾀﾃ ■■
110 WIDTH80,25:N=50:R=5:L=10:P=12
120 PRINTCHR$(12):FOR I=0 TO N-1
130 LOCATE P*(I¥L),I MOD L
140 PRINTUSING"## ";I+1
150 NEXT:LOCATE0,L+1:END
```

となります。
また、同じように、横方向に、1，2，3……と表示したいとき、

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40
41	42	43	44	45
46	47	48	49	50

プログラムは、

```
100 ' ■■ ヨコ ■■
110 WIDTH80,25:N=50:R=5:L=10:P=12
120 PRINTCHR$(12):FOR I=0 TO N-1
130 LOCATE P*(I MOD R),I¥R
140 PRINTUSING"## ";I+1
150 NEXT:LOCATE0,L+1:END
```

となります。

表示したい数値や文字列を配列変数に入れておけば、これを使って、簡単に表示することができます。

2-4.GET@、PUT@の使い方

GET@、PUT@には、画面上の任意の範囲のキャラクタまたはドットグラフィックスを、

GET@………配列へ読み込む

PUT@………配列の内容を画面へ表示するものです。

という機能があります。

GET@、PUT@には、次の3種類のモードがあります。

1. 通常モード(属性なし、キャラクタモード)
2. Gオプションつき(属性なし、グラフィックモード)

   GET(X,Y)−(x,y)、<配列名>、G

   └─アルファベットのGまたはgをつける。

3. Aオプションつき(属性つき混在モード)

配列に読み込むとき、色、リバース、ブリンクなどの属性(アトリビュート)も一緒に読み込むものです。

GET@A……属性も含めて配列へ読み込む。

PUT@A……属性も含めて配列の内容を画面へ表示する。

└─アルファベットのAまたはaをつける。

〔図表2-8〕

アトリビュート	グラフィック キャラクタ	文	座標の範囲	必要な配列の大きさ(切上げ)
なし	キャラクタ	GET@(X,Y)−(x,y),〈配列名〉 PUT@(X,Y)−(x,y),〈配列名〉〔,〈条件1〉〕	x…0～1行の桁数−1 y…0～画面の行数−1	キャラクタ数／M
	ドット グラフィクス	GET@(X,Y)−(x,y),〈配列名〉, G PUT@(X,Y)−(x,y),〈配列名〉,〈条件2〉	x…0～(1行の桁数)＊2−1 y…0～(画面の行数)＊4−1	(ドット数／8＋2)／M
つき	キャラクタ ドットグラフィクス 混在可	GET@A(X,Y)−(x,y),〈配列名〉 PUT@A(X,Y)−(x,y),〈配列名〉	x…0～1行の桁数−1 y…0～画面の行数−1	キャラクタ数＊2／M

アトリビュートつきとは、カラー指定やリバース・ブリンクなどを含めて、配列へ読み込み(GET@A…)画面へ表示(PUT@A…)をするものです。

〈条件1〉 0～7又は8～15の数値

		白黒モード	カラーモード
0	8	ノーマル	黒
1	9	シークレット	青
2	10	ブリンク	赤
3	11	シークレット	紫
4	12	リバース	緑
5	13	リバースシークレート	水色
6	14	リバースブリンク	黄
7	15	リバースシークレート	白

白黒モード

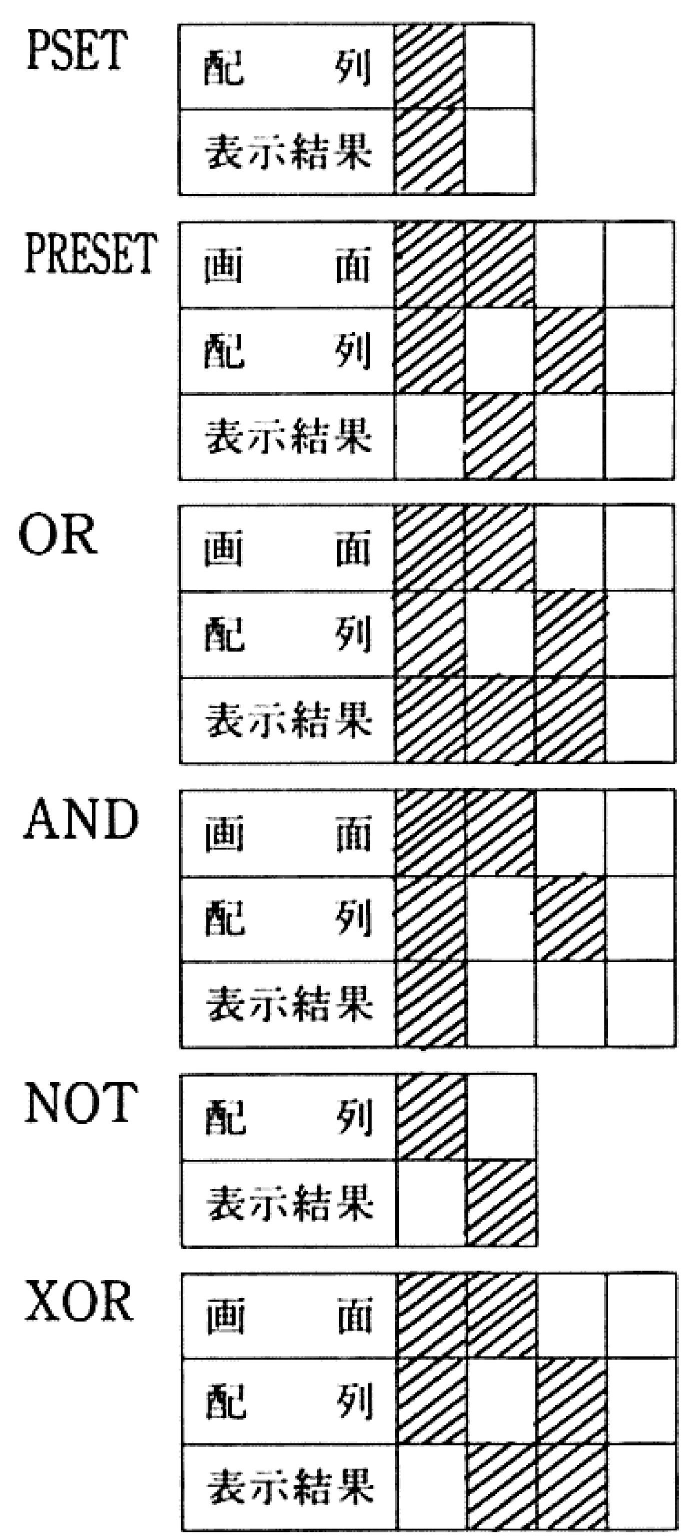
3	2	1	0
――	リバース	ブリンク	シークレット

カラーモード

3	2	1	0
――	緑	赤	青

★条件を省略した場合は、白黒モードの場合ノーマル、カラーモードでは、白となります。
★変数も使えます。

〈条件2〉



＊省略はできません。

xの最大値

桁数		キャラクタ	ドット グラフィクス
36		35	71
40	カラー	38	77
	白黒	39	79
72		71	143
80	カラー	78	157
	白黒	79	159

yの最大値

行数	キャラクタ	ドットグラフィクス
20	19	79
25	24	99

M…変数の型を示す数値
整数型(%)…2
単精度型(!)…4
倍精度型(#)…8
配列は、0から始まるので、DIM文で宣言するのは、この式による計算値から1引いたもので良い。

2-4-1 配列の宣言

配列は、あらかじめDIM文により宣言しておく必要があります。配列の大きさが10以下であっても、前に、その配列を使用していなければ、宣言する必要があります。

2-4-2 配列の大きさ

配列の大きさは、読み込むキャラクタ数、ドット数、Aオプションつきかどうかで異なり、少なくとも表に示す値以上とする必要があります。

2-4-3 配列変数の型

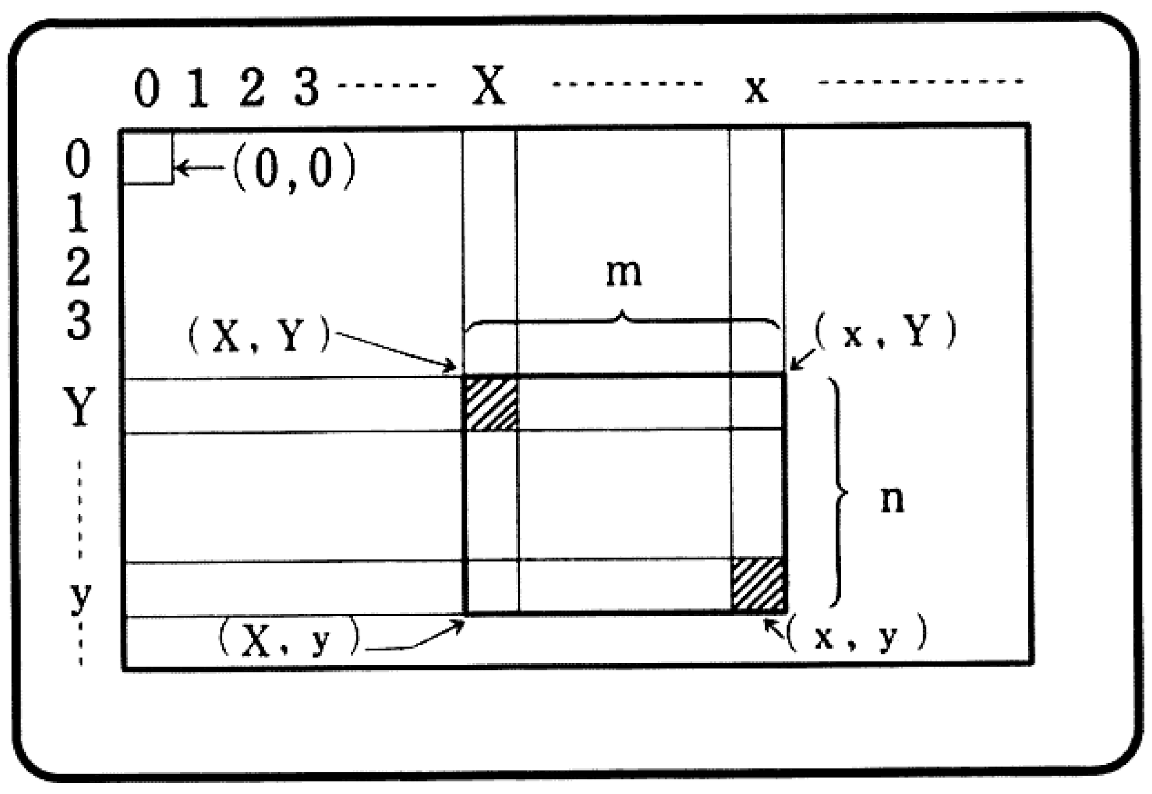
変数の型には、整数型(%)、単精度型(!)、倍精度型(#)の3種類ありますが、メモリ効率上から、整数型(%)を用いるのがよいでしょう。

例えば、倍精度変数を用いると、8バイト単位となり、キャラクタ数やドット数により1～7バイト無駄にメモリを使う場合がありますが、整数型だと、2バイト単位ですので無駄になるとしても1バイトで済みます。

2-4-4 範囲の指定

画面の左上隅を(0,0)水平方向をx座標垂直方向をy座標として指定します。

〔図表2-9〕



一般に、

(X, Y)−(x, y)

と指定しますが、

(x, y)−(X, Y)

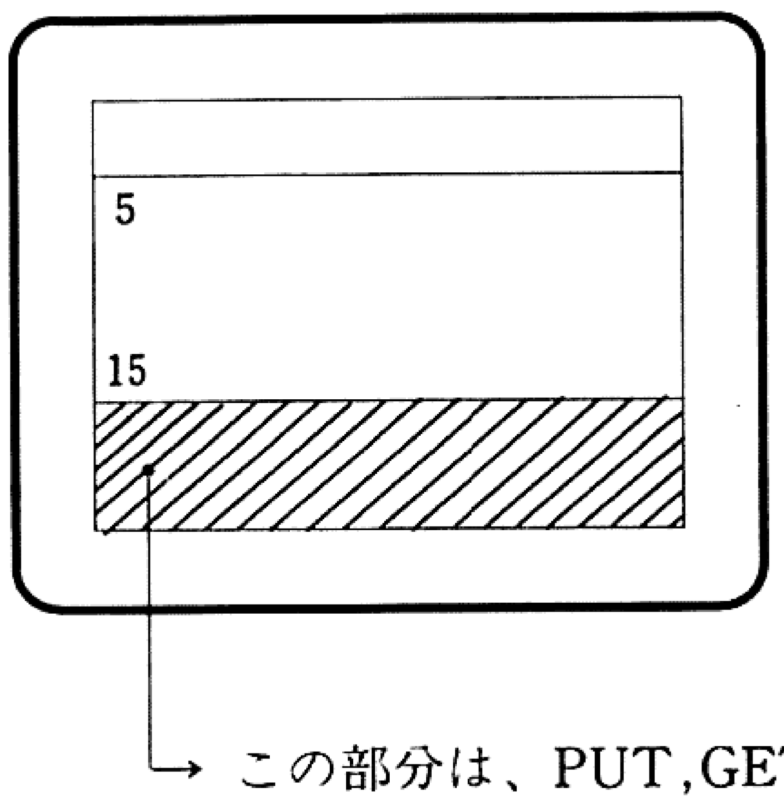
(x, Y)−(X, y)

(X, y)−(x, Y)

などと、指定しても、何ら変わるところは、ありません。

また、PUT @するとき、GET @したときの縦横の数を変えなければ、どこへPUT @しても、同じものが表示されます。縦横の数が変わっても、縦横を掛け合わせた数が、GET @したときの数よりも等しいか小さければ、形を変えて表示されます。

CONSOLE文(例えばCONSOLE5, 11)を実行後は、スクロール範囲の画面下側は、PUT, GETできません。



この部分は、PUT,GETできません。

```
100 CONSOLE0,25,0,0:DIM A%(8)
110 PRINTCHR$(12)"@ABCDE":PRINT"FGHIJK":PRINT"LMNOPQ"
120 GET@(0,0)-(5,2),A%
130 PUT@(0,5)-(5,7),A%
140 PUT@(9,5)-(17,6),A%
150 PUT@(21,5)-(23,10),A%
160 PUT@(27,5)-(29,7),A%
200 LOCATE0,12
```

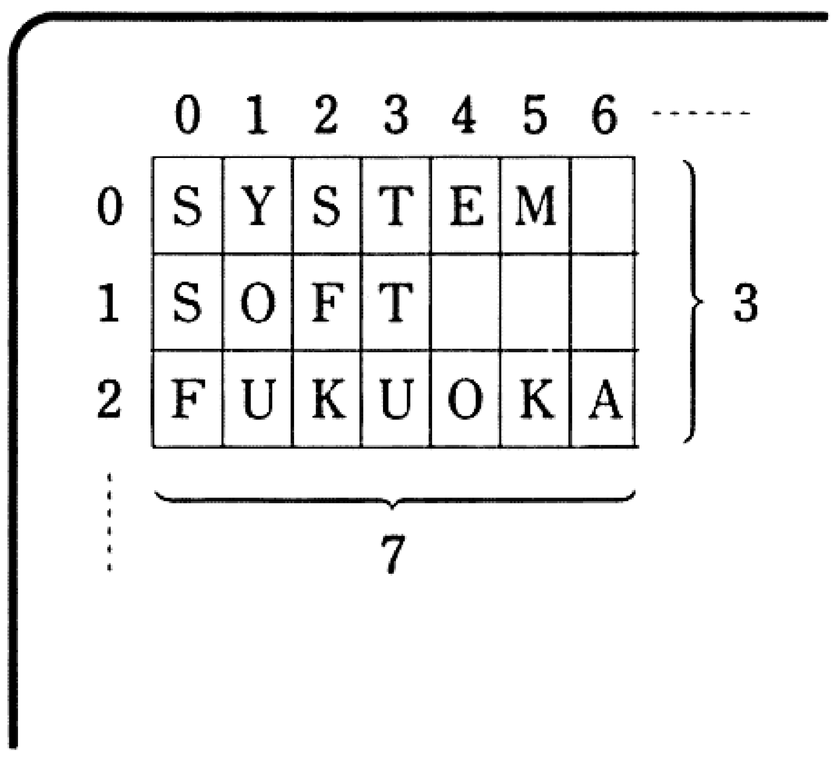
```
@ABCDE     @ABCDEFGH    @AB    @AB
FGHIJK     IJKLMNOPQ    CDE    CDE
LMNOPQ                  FGH    FGH
                        IJK
                        LMN
                        OPQ

OK
```

2-4-5 画面表示と配列の関係

それでは、画面の内容が、配列へ、どのように読み込まれているか調べてみましょう。

〔図表2-10〕



(1)キャラクタモードの場合

画面左上に、右図のように表示したものを読み込んで調べてみましょう。

GET @する範囲……(0,0)－(6,2)

3×7＝21キャラクタ

配列の大きさ(整数型の場合)……………

21／2＝10.5→11

配列は0から始まるので配列の宣言は、

11－1→DIM A%(10)以上を宣言します。

次のプログラムで配列を16進数表示します。

```
100 CONSOLE0,25,0,0:DIM A%(10)
110 PRINTCHR$(12)"SYSTEM":PRINT"SOFT":PRINT"FUKUOKA"
120 GET@(0,0)-(6,2),A%
130 PRINT:FORI=0TO10
140 PRINTUSING"A%(##)=&h";I;:PRINTHEX$(A%(I)):NEXT:PRINT
```

```
SYSTEM
SOFT
FUKUOKA

A%( 0)=&h5953
A%( 1)=&h5453
A%( 2)=&h4D45
A%( 3)=&h5320
A%( 4)=&h464F
A%( 5)=&h2054
A%( 6)=&h2020
A%( 7)=&h5546
A%( 8)=&h554B
A%( 9)=&h4B4F
A%(10)=&h41

Ok
a%(6)=&he4e5
Ok
put@(20,0)-(26,2),a%
Ok
```

```
SYSTEM
SOFT
FUKUOKA
```

キャラクターコード表より──→

5 9……Y	5 3……S
5 4……T	5 3……S
4 D……M	4 5……E
5 3……S	0 0……␣→ヌルストリング（スペース）
4 6……F	4 F……O
0 0……␣	5 4……T
0 0……␣	0 0……␣
5 5……U	4 6……F
5 5……U	4 B……K
4 B……K	4 F……O
0 0……␣	4 1……A

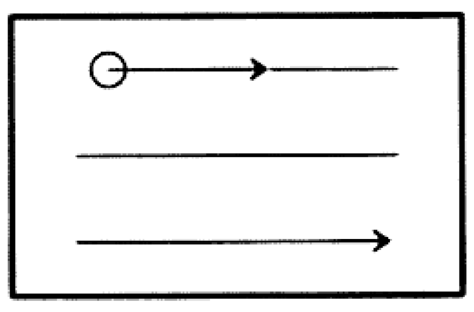
5 3 0 0→5 3 2 0

5 4　　→2 0 5 4

0　　　→2 0 2 0

COLOR, &H20：PRINT CHR$(12)をしておくとスペースの部分は、20Hとなる。

A%(0)の下位1バイトに"S"、上位1バイトに"Y"というように、画面の左上から順にスペースも含めて、1数値に2バイト(整数型)づつ読み込まれていることがわかります。

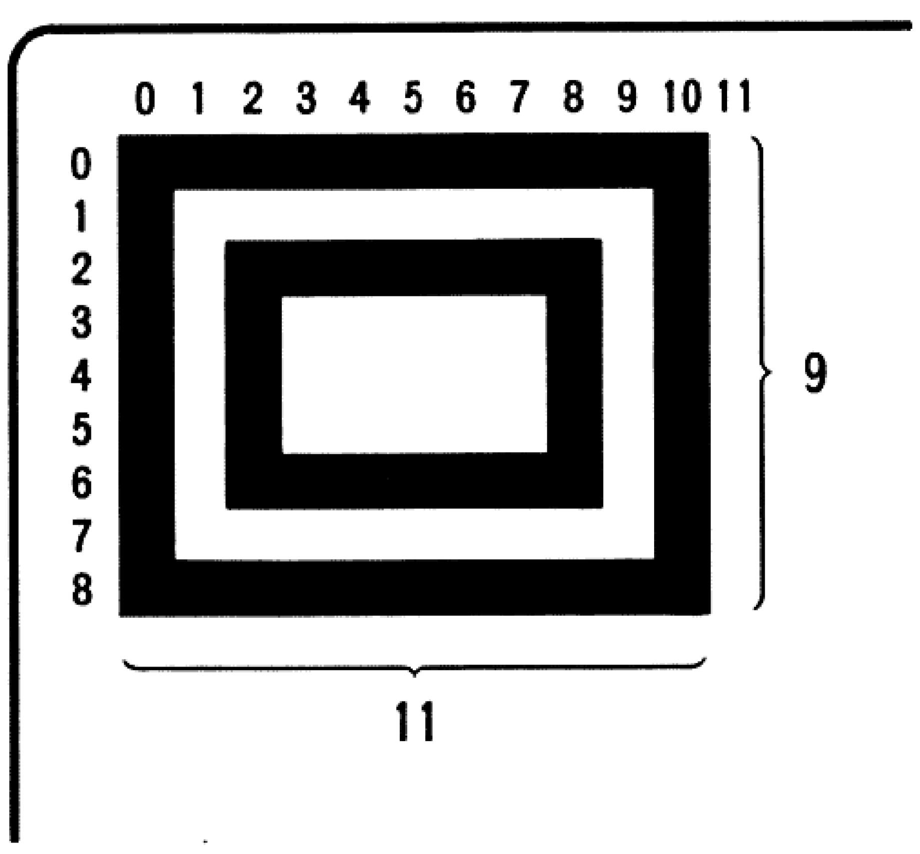


配列変数に表示したいキャラクタコードを代入して、PUT @で表示してみてください。

(2)ドットグラフィクスモードの場合

画面左上に右図のように表示したものを読み込んで調べてみましょう。

〔図表2-11〕



ドットグラフィクスなのでGオプションとします。

GET @する範囲は、(0,0)−(10,8)

11×9＝99ドット

配列の大きさ(整数型の場合)…………………

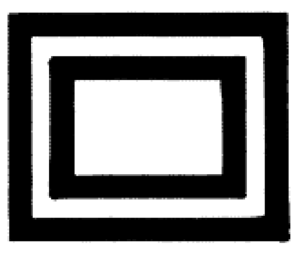
(99/8＋2)/2＝7.5→8

配列は、0から始まるので8−1→DIM A%(7)以上を宣言します。

次のプログラムで配列を16進数表示してみましょう。

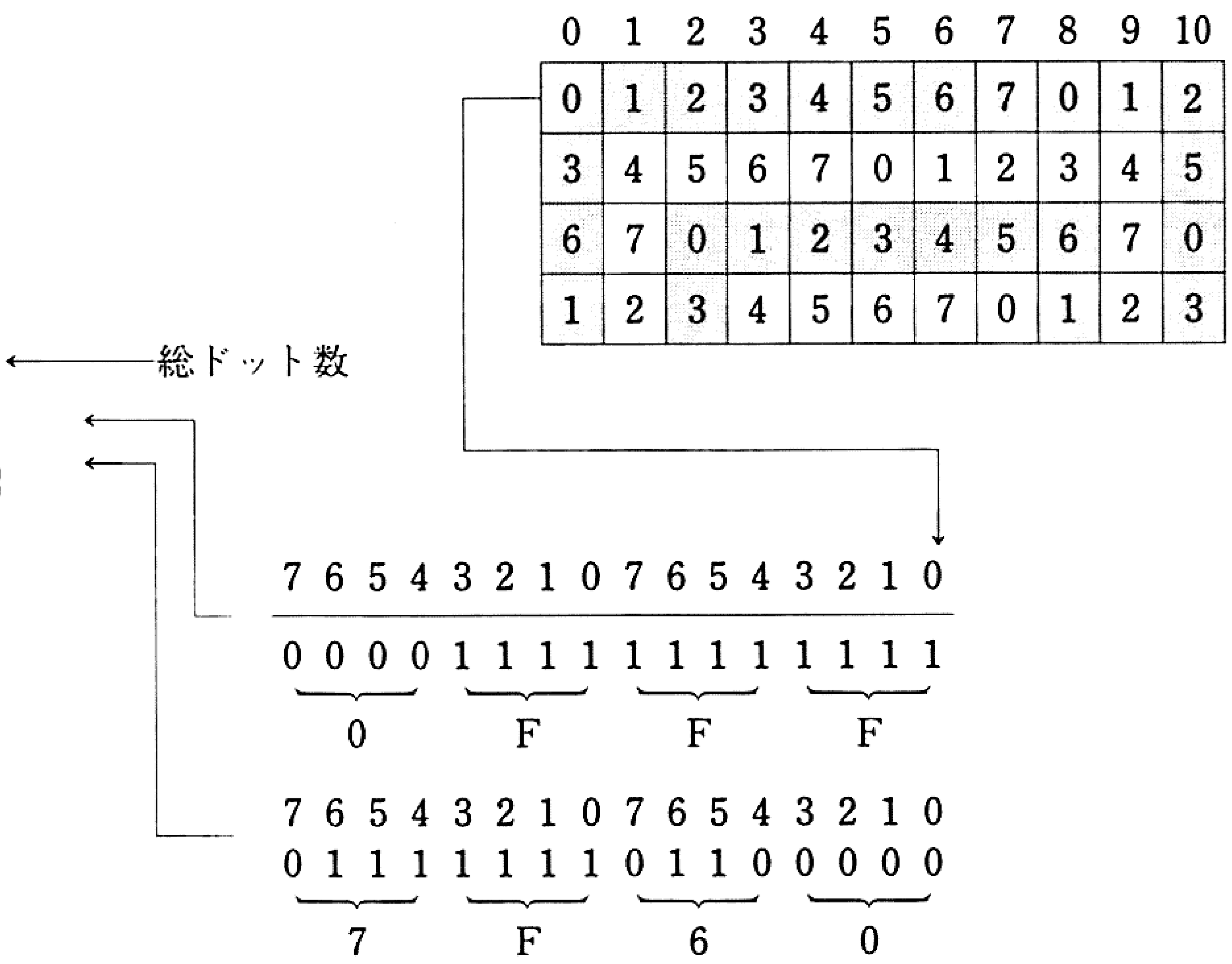
```
100 CONSOLE0,25,0,0:DIMA%(7)
110 PRINTCHR$(12);
120 LINE(0,0)-(10,8),PSET,B
130 LINE(2,2)-(8,6),PSET,BF
140 GET@(0,0)-(10,8),A%,G
150 LOCATE0,5:FORI=0TO7
160 PRINTUSING"A%(#)=&h&   &=";I,HEX$(A%(I));
170 PRINTA%(I):NEXT:PRINT
```

実行結果は、次のようになります。



```
A%(0)=&h63   = 99
A%(1)=&hFFF  = 4095
A%(2)=&h7F60= 32608
A%(3)=&hDBFB=-9221
A%(4)=&hFEDF=-289
A%(5)=&h37F6= 14326
A%(6)=&hFF80=-128
A%(7)=&h7    = 7

Ok
```



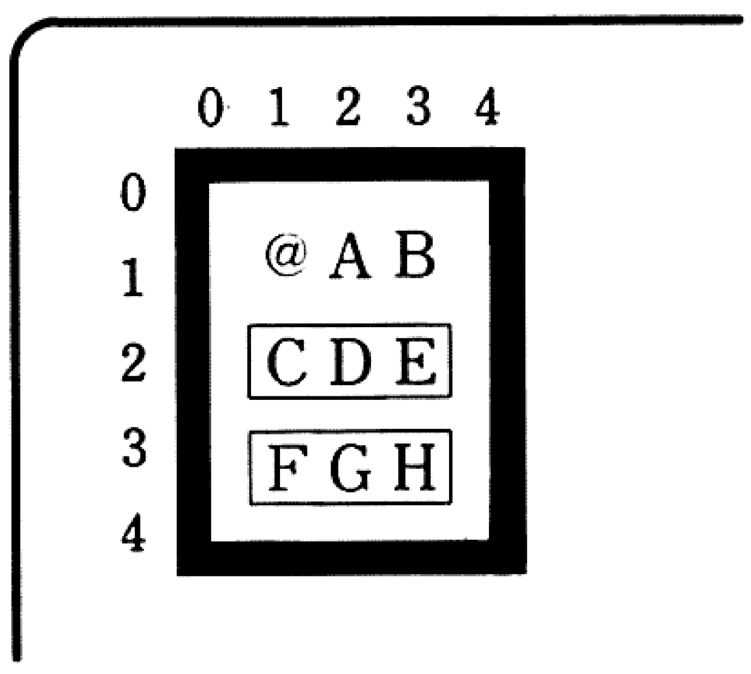
A％(0)には、ドット総数がはいり、A％(1)以降には、画面のドットを左上から第0ビットとし、ドットがある部分を1、ない部分を0に対応させた2進数を、16ビット毎に16進数とした値が入っています。

(3)アトリビュートモードの場合

（カラーやドットグラフィクス、ブリンクなどを含むとき）

画面左上に、右図のようにキャラクタとドットグラフィクスで表示したものを読み込んで調べてみましょう。

〔図表2-12〕



ドットグラフィクスやカラー指定を含んで読み込む場合、Aオプションとなります。

GET＠Aする範囲(キャラクタ単位で指定)

(0,0)−(4,4)

5×5＝25キャラクタ

配列の大きさ(整数型の場合)……25×2/2＝25

配列は、0から始まるので25－1→DIM A％(24)以上を宣言します。

次のプログラムで配列の内容を見てみましょう。

```
100 CONSOLE0,25,0,0:WIDTH80,25:DIMA%(24):PRINTCHR$(12)
110 COLOR2:PRINT" @AB":COLOR4:PRINT" CDE":COLOR6:PRINT" FGH"
120 COLOR0: LINE(0,0)-(9,19),PSET,B
130 GET@A(0,0)-(4,4),A%
140 PUT@A(20,3)-(24,7),A%
150 LOCATE0,9:FORI=0TO24:PRINTUSING"A%(##)=&h";I;
160 PRINTHEX$(A%(I)),:NEXT:PRINT:PRINT
170 PRINT"A%(   )=&h"STRING$(6,CHR$(29));
180 LINEINPUTK$:K=VAL(K$)
190 LOCATE9,CSRLIN-1:LINEINPUTK$:A%(K)=VAL("&h"+K$):GOTO140
```

これを実行すると、

```
@AB
CDE
FGH

          @AB
          CDE
          FGH
```

```
A%( 0)=&h801F A%( 1)=&h8011 A%( 2)=&h8011 A%( 3)=&h8011 A%( 4)=&h80F1
A%( 5)=&h800F A%( 6)=&h240  A%( 7)=&h241  A%( 8)=&h242  A%( 9)=&h80F0
A%(10)=&h800F A%(11)=&h443  A%(12)=&h444  A%(13)=&h445  A%(14)=&h80F0
A%(15)=&h800F A%(16)=&h646  A%(17)=&h647  A%(18)=&h648  A%(19)=&h80F0
A%(20)=&h808F A%(21)=&h8088 A%(22)=&h8088 A%(23)=&h8088 A%(24)=&h80F8

A%(   )=&h
```

この表示を、整数型配列変数に読み込んだ場合、次のような構成となります。

〔図表2-13〕

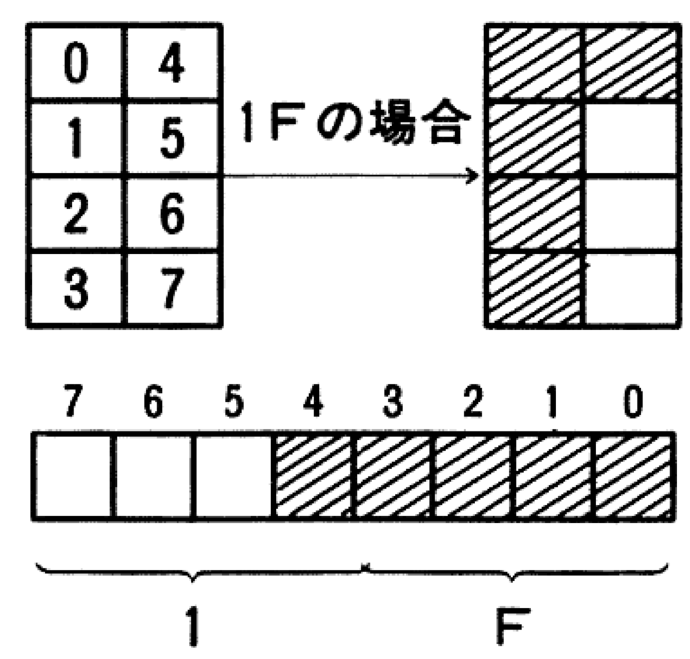
(下位)キャラクタコード
または、グラフィックコード

A%(0)=&H801F

(上位)アトリビュートコード

アトリビュートコードは、V-RAMのアトリビュートエリアと同じ表現であり、上の例では、白黒モードで、80H(第7ビットのみ1)なので、グラフィックモードということになる。

アトリビュートコードがグラフィックモードのとき、画面の1キャラクタ分は、下図のように8ドットに分割され、8ビットコードとして、V-RAMや、GET@した時の配列内へ格納されます。



白黒モードのとき

7	6	5	4	3	2	1	0
G/C	―	アンダライン	アッパライン	―	リバース	ブリンク	シークレット

カラーモードのとき

7	6	5	4	3	2	1	0
―	―	アンダライン	アッパライン	0	リバース	ブリンク	シークレット

↕ このビットが0か1かで2通りある。

7	6	5	4	3	2	1	0
緑	赤	青	G/C	1	―	―	―

＊G…グラフィックモード
C…キャラクタモード

A%(16)=&H646

00000110で キャラクタモード・リバース・ブリンクを表わしている。

キャラクタコードで"F"を表わしている。

すなわち、データとアトリビュートが1バイトづつ格納されており、ふつうのキャラクタモードの2倍の配列が必要になるわけです。

前記プログラムで、A%(_)=&H____に適当な配列No.と16進データを入れて、右側のグラフィック表示がどのように変わるか試してください。

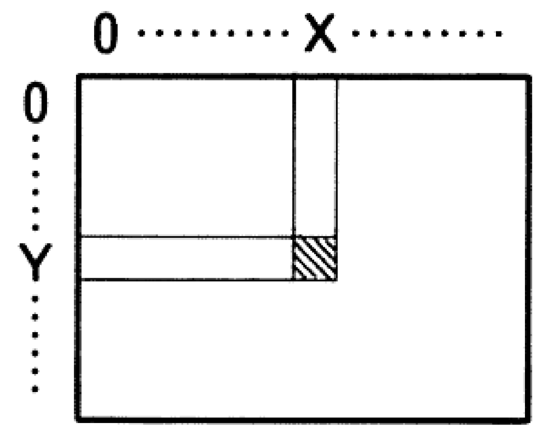
また、行番号100および120を、

```
100 CONSOLE0,25,,1
120 COLOR7:LINE………
```

と変更し、カラーモードについても試してください。

2-4-6 画面任意の位置の色を判定する方法

GET＠A……が配列に読み込む数値から、画面の任意の座標（キャラクタ単位）の色やその他の情報を知ることができます。



GET＠A(X,Y)−(X,Y), A％

を実行すると、A％(0)の2バイト(16ビット)のデータは、例えば、カラーモードのとき、次のような構成になっています。

〔図表12-14〕

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
緑	赤	青	G/C	1	—	—	—								

ビット15〜8：アトリビュートコード　格納アドレス VARPTR(A％(0))＋1

ビット7〜0：画面のキャラクタコード または、グラフィックコード　VARPTR(A％(0))

したがって、例えば、表示色を知りたいときは、その情報が上位3ビットにあたるので、

C＝PEEK(VARPTR(A％(0))＋1)¥32

とすれば、カラーコードの0〜7がCに入ります。

また、

G＝A％(0)AND &H1000

とすると、

G＝0のときキャラクタモード

G<>0のときグラフィックモード

となります。

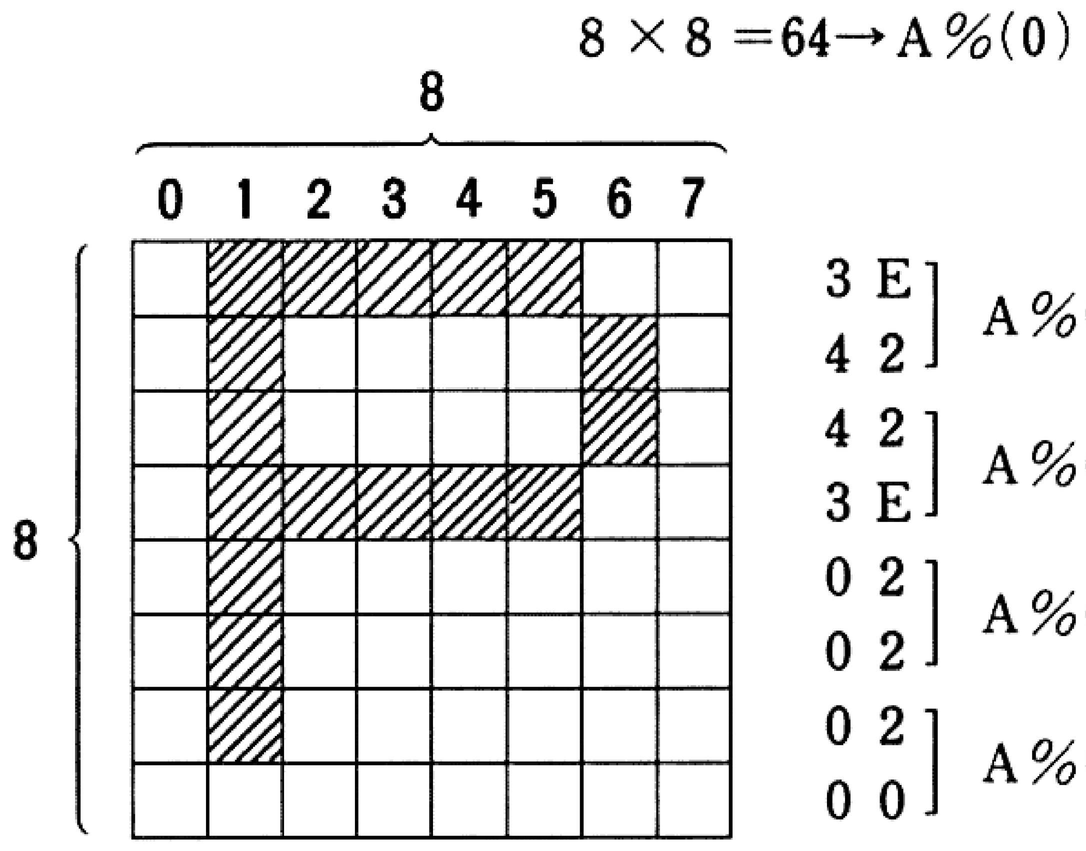
2-4-7 画面に大きくキャラクタを表示する方法

画面に、8×8のドットグラフィクスでキャラクタを表示しようとする場合、右図のように8×8のマス目に表示したいパターンを描き、それを配列にします。

大文字の"P"の場合、次のようになります。

A％(0)＝64……ドット数
A％(1)＝&H423E
A％(2)＝&H3E42
A％(3)＝&H0202
A％(4)＝&H0002

〔図表12-15〕



注）下位・上位の順を間違えないようにすること。

これを、表示したい任意の位置(X, Y)にPUTすれば良いわけです。

PUT @(X, Y)−(X＋7, Y＋7), A%, PSET

パターンデータを2次元配列A%(3,200) などに読み込み、必要なときに、

B%(0)=64：FOR i＝0 to 3：B%(i＋1)＝A%(i,j)：NEXT

PUT @(X, Y)−(X＋7, Y＋7), B%, PSET

として表示するのも一つの方法でしょう。

下覧に、キャラクタのパターンデータ（16進数）をリストしておきますので、利用してください。

〔図表12-16〕キャラクタフォントデータ表　　　　**DEFINT A：A(0)=64**

ASCII		a(1)	a(2)	a(3)	a(4)
20		0000	0000	0000	0000
21	!	1010	1010	0000	0010
22	"	2424	0024	0000	0000
23	#	2424	247E	247E	0024
24	$	7810	3814	3C50	0010
25	%	4600	1026	6408	0062
26	&	120C	0C12	2252	005C
27	'	1020	0008	0000	0000
28	(	1020	0808	1008	0020
29	)	0804	1010	0810	0004
2A	*	5410	FE38	5438	0010
2B	+	1000	7C10	1010	0000
2C	,	0000	0000	1000	0810
2D	-	0000	7E00	0000	0000
2E	.	0000	0000	1800	0018
2F	/	4000	1020	0408	0002
30	0	423C	5A62	4246	003C
31	1	1810	1014	1010	007C
32	2	423C	3040	020C	007E
33	3	423C	3840	4240	003C
34	4	3020	2428	207E	0020
35	5	027E	201E	2240	001C
36	6	0438	3E02	4242	003C
37	7	427E	1020	0808	0008
38	8	423C	3C42	4242	003C
39	9	423C	7C42	2040	001C
3A	:	0000	0010	1000	0000
3B	;	0000	0010	1000	0810
3C	<	1870	060C	180C	0070
3D	=	0000	007E	007E	0000
3E	>	180E	6030	1830	000E
3F	?	423C	3040	0008	0008
40	@	4438	6A52	0432	0078
41	A	2418	7E42	4242	0042
42	B	443E	3C44	4444	003E
43	C	4438	0202	4402	0038
44	D	241E	4444	2444	001E
45	E	017F	3F01	0101	007F
46	F	017F	3F01	0101	0001
47	G	4438	7202	4442	0038
48	H	4242	7E42	4242	0042
49	I	1038	1010	1010	0038
4A	J	2070	2020	2220	001C
4B	K	2242	0E12	2212	0042
4C	L	0202	0202	0202	007E
4D	M	6642	5A5A	4242	0042
4E	N	4642	524A	4262	0042
4F	O	2418	4242	2442	0018
50	P	423E	3E42	0202	0002
51	Q	2418	4242	2452	0058
52	R	423E	3E42	2212	0042
53	S	423C	3C02	4240	003C
54	T	107C	1010	1010	0010
55	U	4242	4242	4242	003C

ASCII		a(1)	a(2)	a(3)	a(4)
56	V	4242	2442	1824	0018
57	W	4242	5A42	665A	0042
58	X	4242	1824	4224	0042
59	Y	4444	3844	1010	0010
5A	Z	407E	1820	0204	007E
5B	[	043C	0404	0404	003C
5C	¥	4444	7C28	7C10	0010
5D	]	203C	2020	2020	003C
5E	^	2810	0044	0000	0000
5F	_	0000	0000	0000	007E
60	`	0000	0000	0000	0000
61	a	0000	203C	223C	005C
62	b	0202	463A	4642	003A
63	c	0000	423C	4202	003C
64	d	4040	625C	6242	005C
65	e	0000	423C	027E	003C
66	f	4830	3E08	0808	0008
67	g	0000	625C	5C62	3C40
68	h	0202	463A	4242	0042
69	i	0010	1018	1010	0038
6A	j	0020	2030	2020	1C22
6B	k	0202	1222	160A	0022
6C	l	1018	1010	1010	0038
6D	m	0000	926E	9292	0092
6E	n	0000	463A	4242	0042
6F	o	0000	423C	4242	003C
70	p	0000	463A	3A46	0202
71	q	0000	625C	5C62	4040
72	r	0000	463A	0202	0002
73	s	0000	027C	403C	003E
74	t	0808	083E	4808	0030
75	u	0000	4242	6242	005C
76	v	0000	4242	2442	0018
77	w	0000	9292	9292	006C
78	x	0000	2442	2418	0042
79	y	0000	4242	5C62	3C40
7A	z	0000	207E	0418	007E
7B	{	0870	0408	0808	0070
7C	¦	1010	0000	1000	0010
7D	}	100E	2010	1010	000E
7E	~	920C	0060	0000	0000
7F		0000	0000	0000	0000
80	▁	0000	0000	0000	FF00
81	▂	0000	0000	0000	FFFF
82	▃	0000	0000	FF00	FFFF
83	▄	0000	0000	FFFF	FFFF
84	▅	0000	FF00	FFFF	FFFF
85	▆	0000	FFFF	FFFF	FFFF
86	▇	FF00	FFFF	FFFF	FFFF
87	█	FFFF	FFFF	FFFF	FFFF
88	▏	0101	0101	0101	0101
89	▎	0303	0303	0303	0303
8A	▍	0707	0707	0707	0707
8B	▌	0F0F	0F0F	0F0F	0F0F

ASCII		a(1)	a(2)	a(3)	a(4)
8C	▋	1F1F	1F1F	1F1F	1F1F
8D	▊	3F3F	3F3F	3F3F	3F3F
8E	▉	7F7F	7F7F	7F7F	7F7F
8F	┼	1010	1010	10FF	1010
90	┴	1010	1010	00FF	0000
91	┬	0000	0000	10FF	1010
92	┤	1010	1010	101F	1010
93	├	1010	1010	10F0	1010
94	▔	00FF	0000	0000	0000
95	─	0000	0000	00FF	0000
96	│	1010	1010	1010	1010
97	▕	8080	8080	8080	8080
98	┌	0000	0000	10F0	1010
99	┐	0000	0000	101F	1010
9A	└	1010	1010	00F0	0000
9B	┘	1010	1010	001F	0000
9C	╭	0000	0000	20C0	1010
9D	╮	0000	0000	0403	0808
9E	╰	1010	C020	0000	0000
9F	╯	0808	0304	0000	0000
A0		0000	0000	0000	0000
A1	｡	0000	0000	141C	001C
A2	｢	0838	0808	0000	0000
A3	｣	0000	1000	1010	001C
A4	､	0000	0000	0804	0010
A5	･	0000	1800	0018	0000
A6	ｦ	7E00	7E40	2040	001C
A7	ｧ	0000	407C	1030	0008
A8	ｨ	0000	2040	2830	0020
A9	ｩ	0000	7C10	4044	0030
AA	ｪ	0000	7C00	1010	007C
AB	ｫ	0000	7C20	2830	0024
AC	ｬ	0000	7C08	2848	0008
AD	ｭ	0000	3800	2020	007C
AE	ｮ	0000	407C	407C	007C
AF	ｯ	0000	5400	4054	0030
B0	ｰ	0000	7C00	0000	0000
B1	ｱ	407E	2840	0818	0004
B2	ｲ	2040	1810	1214	0010
B3	ｳ	7E10	4242	2040	001C
B4	ｴ	7C00	1010	1010	007C
B5	ｵ	7E10	1810	1214	0010
B6	ｶ	7E08	4848	4848	0024
B7	ｷ	7C10	7C10	1010	0010
B8	ｸ	4478	2042	0810	0006
B9	ｹ	7C04	1012	1010	0008
BA	ｺ	7E00	4040	4040	007E
BB	ｻ	7E24	2424	1020	0008
BC	ｼ	0600	4600	2040	001E
BD	ｽ	407E	1020	2418	0042
BE	ｾ	7F02	2242	0202	007C
BF	ｿ	4242	4044	1020	000C
C0	ﾀ	4478	2846	0810	0006
C1	ﾁ	2000	101C	107E	0E10

ASCII		a(1)	a(2)	a(3)	a(4)
C2	ﾂ	4A00	4A4A	1020	000E
C3	ﾃ	003C	107E	0810	0004
C4	ﾄ	0808	1808	0828	0008
C5	ﾅ	1010	107E	0810	0004
C6	ﾆ	3C00	0000	0000	007E
C7	ﾇ	7C00	2840	2810	0044
C8	ﾈ	7C10	1020	5438	0010
C9	ﾉ	4040	2040	0810	0004
CA	ﾊ	1008	4020	4242	0042
CB	ﾋ	0202	027E	0202	007C
CC	ﾌ	7E00	4040	1020	0008
CD	ﾍ	0800	2214	4040	0000
CE	ﾎ	7C10	1010	5454	0010
CF	ﾏ	407E	2040	0814	0010
D0	ﾐ	3C00	3C00	3E00	0040
D1	ﾑ	0810	2204	7E42	0040
D2	ﾒ	4000	2840	2810	0006
D3	ﾓ	7E00	7E08	0808	0078
D4	ﾔ	0404	447E	0404	0004
D5	ﾕ	3C00	2020	2020	007E
D6	ﾖ	407E	7E40	4040	007E
D7	ﾗ	003C	407E	1020	0008
D8	ﾘ	4242	4242	2040	0018
D9	ﾙ	1414	5414	3454	0012
DA	ﾚ	0404	4404	1424	000C
DB	ﾛ	7E00	4242	4242	007E
DC	ﾜ	7E00	4242	2040	0018
DD	ﾝ	0E00	4040	1020	000E
DE	ﾞ	1208	0004	0000	0000
DF	ﾟ	0A0E	000E	0000	0000
E0	═	0000	00FF	FF00	0000
E1	╞	1010	10F0	F010	1010
E2	╪	1010	10FF	FF10	1010
E3	╡	1010	101F	1F10	1010
E4	◢	C080	F0E0	FCF8	FFFE
E5	◣	0301	0F07	3F1F	FF7F
E6	◥	FEFF	F8FC	E0F0	80C0
E7	◤	7FFF	1F3F	070F	0103
E8	♠	3810	FE7C	38FE	007C
E9	♥	FE6C	FEFE	7CFE	1038
EA	♦	3810	FE7C	387C	0010
EB	♣	3838	FEFE	10D6	007C
EC	●	3C00	7E7E	7E7E	003C
ED	○	3C00	4242	4242	003C
EE	╱	4080	1020	0408	0102
EF	╲	0201	0804	2010	8040
F0	╳	4281	1824	2418	8142
F1	円	92FE	FE92	8282	0082
F2	年	7E02	3C12	7E14	0010
F3	月	84FC	84FC	84FC	0082
F4	日	427E	7E42	4242	007E
F5	時	7720	F725	F745	4050
F6	分	1800	4224	283C	0024
F7	秒	485C	E8FE	4ADC	0028

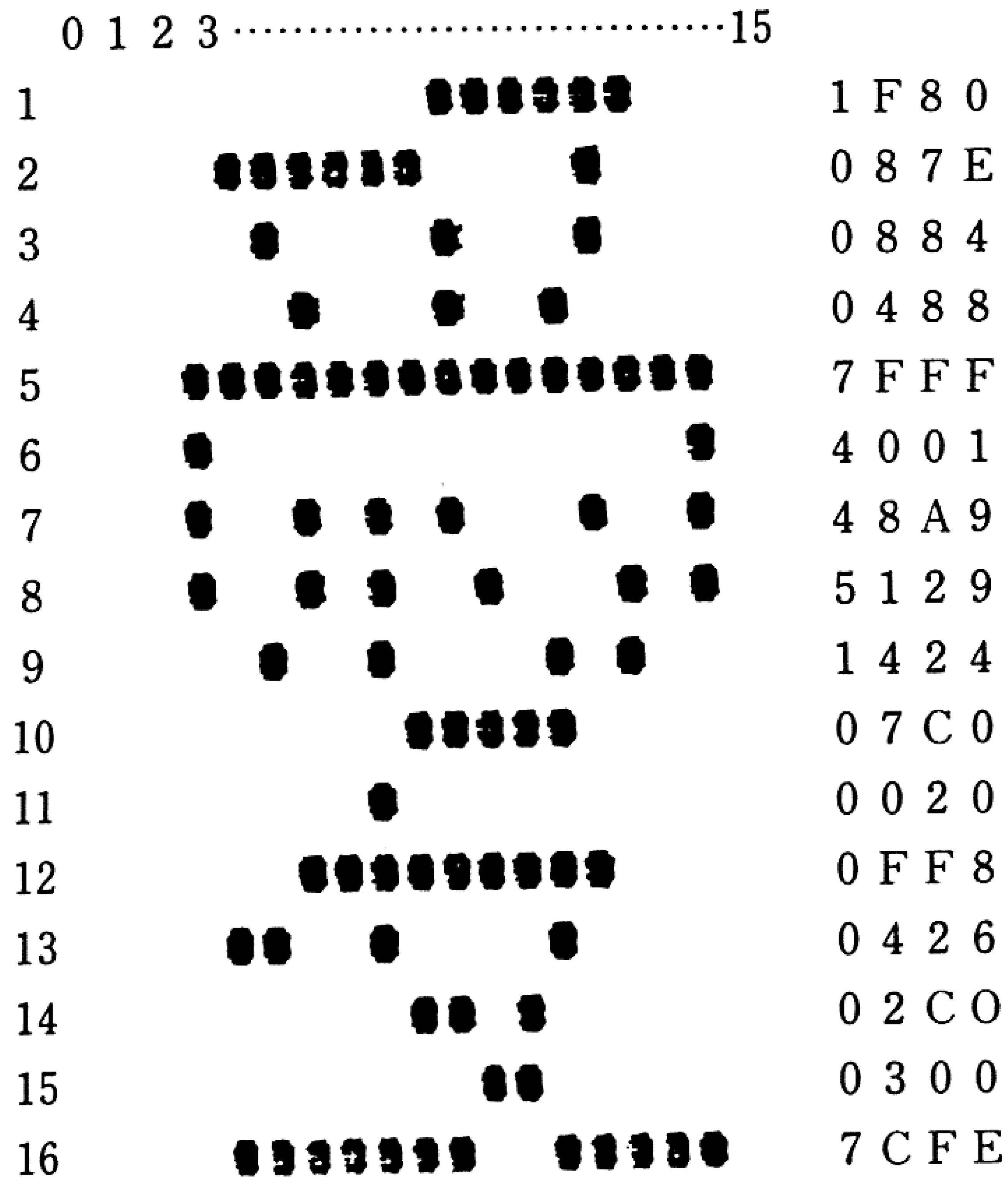
2-4-8 漢字をドット表示する方法

キャラクタの場合と手法は同じですが、漢字の場合、16×16ぐらいのマス目が必要となります。表示したい漢字のパターンを描き、A%(0)に全体のドット数(16×16の場合は256)を入れ、A%(1)から順に左上隅を第0ビットとする16進数を、配列に書き込み、PUT@します。

次のプログラムは、右図のような、〔愛〕という漢字を表示するものです。



〔画面表示後のスクリーンコピー例〕




```
100 DIMA%(16):A%(0)=256:FORI=1TO16:READA$:A%(I)=VAL("&h"+A$):NEXT
110 DATA 1F80,087E,0884,0488,7FFF,4001,48A9,5129
120 DATA 1424,07C0,0020,0FF8,0426,02C0,0300,7CFE
130 PUT@(0,0)-(15,15),A%,PSET
```

2-5. POS, CSRLIN

プログラム実行中に、画面上で、カーソルを移動させる場合、現在位置しているカーソルの水平位置と垂直位置をPOS, CSRLINで知り、その値を増減することで、任意の場所へ移動することができます。

X＝POS（0），Y＝CSRLIN　〔0はダミー〕

LOCATE 10,10：PRINT"ABCDEFG";: X＝POS(0)：Y＝CSRLIN : PRINT X,Y [C/R]

とすると、ABCDEFGの後に17　10が表示されます。文中の ; を取り除いてやると、次の行に 0　11が表示されます。このようにPOS,CSRLIN命令直前の、ステートメントが実行された後のカーソル位置を知らせます。

では、次のプログラムで、カーソルを移動させ、画面の上にカーソルの現在位置を表示させてみましょう。

```
10 PRINT CHR$(12):LOCATE 20,0:PRINT "X=      Y="
20 LOCATE 0,0:X=POS(0):Y=CSRLIN
30 K$=INPUT$(1)
40 IF K$=CHR$(28) THEN X=X+1:IF X>79 THEN X=79
50 IF K$=CHR$(29) THEN X=X-1:IF X<0  THEN X=0
60 IF K$=CHR$(30) THEN Y=Y-1:IF Y<0  THEN Y=0
70 IF K$=CHR$(31) THEN Y=Y+1:IF Y>24 THEN Y=24
80 LOCATE X,Y
100 X1=POS(0):Y1=CSRLIN:LOCATE22,0:PRINTX1:LOCATE29,0:PRINTY1
110 LOCATE X1,Y1
120 GOTO 30
```

ここのCHR$(28)、(29)、(30)、(31) は [→][←][↑][↓] キーのアスキーコードです。このプログラムでは、カーソルの位置（X，Y）に1を加えたり減じたりしていますが、別の数字に置きかえてみると面白い操作ができます。

上のプログラムに画面へ文字を書き込む機能をつけ加えましょう。

```
95 IF K$>CHR$(32) AND K$<CHR$(247) THEN GOSUB 130:GOSUB 140
130 Z=&HF300+120*Y+X:RETURN
140 POKEZ,ASC(K$):X=X+1:LOCATE X,Y:RETURN
```

130行のZ＝&HF300＋120＊Y＋Xは、80桁モードでの現在のカーソル位置(X, Y）に相当するVRAMの位置で、入力コードを直接書き込みます。

更にカーソルを任意の位置に動かし、画面上の文字を抽出する機能を付け加えましょう。

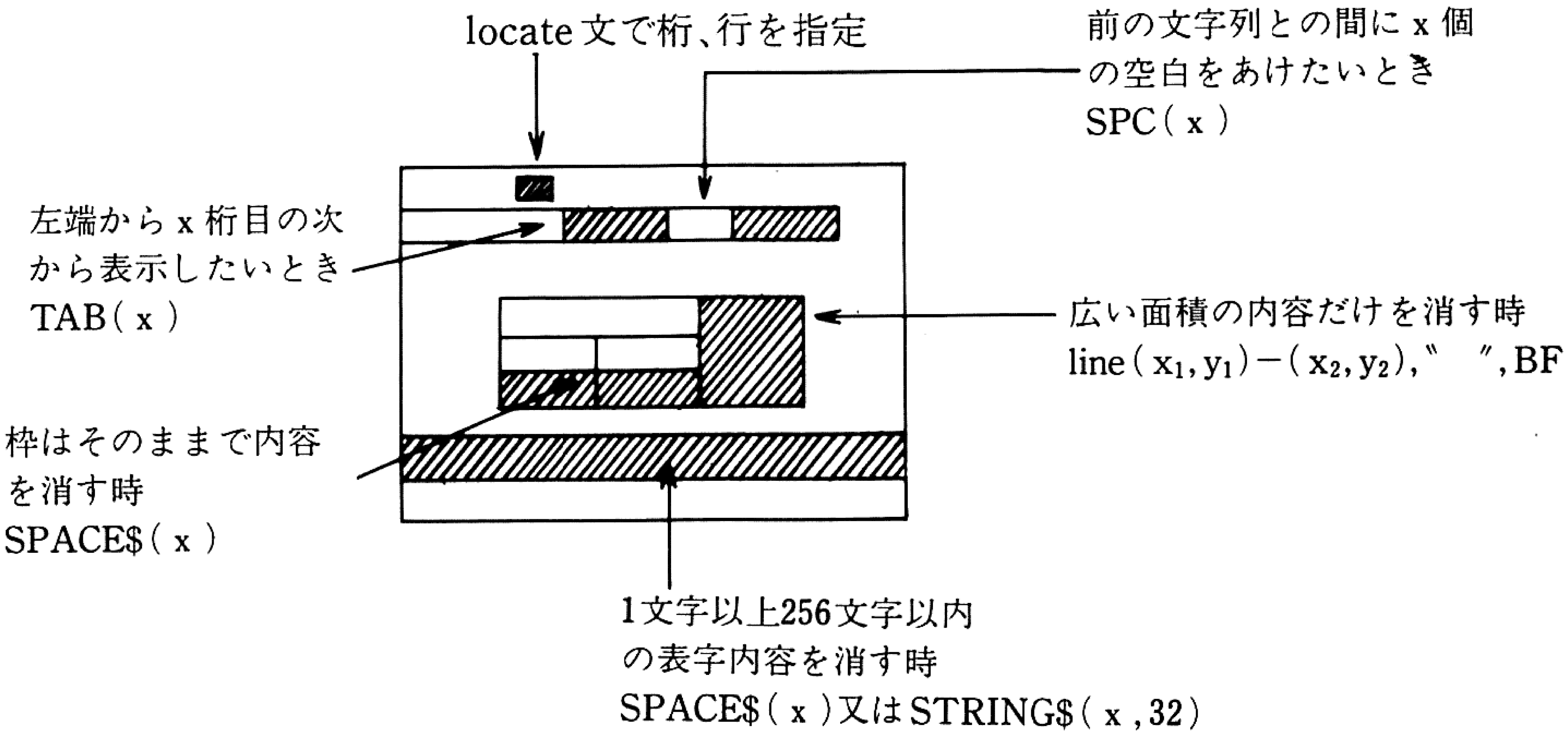
```
98 IF K$=CHR$(13) THEN GOSUB 130:GOSUB 150
150 LOCATE 0,20:PRINT "ﾁｭｳｼﾂ ｽﾙ ﾓｼﾞ ﾉ ｶｽﾞ ﾊ ";:INPUT CN:M$=""
160 FOR I=0 TO CN-1:M$=M$+CHR$(PEEK(Z+I)):NEXT
170 LOCATE 0,21:PRINT SPACE$(80):LOCATE 0,21:PRINT M$:LOCATE X,Y:RETURN
```

RUNさせて下さい。

文字列を書き込んだ後に、カーソルを文字列の中に入れて、RETキーを押し、さらに、抽出したい文字数を入力して再度、RETキーを押すと、望む文字列を抽出できます。

2-6.画面表示と消去

画面の任意の位置に於て、データを入力したり、表示したり、または、画面の一部分を消して、再度記入、表示する場合、LOCATE、TAB、SPC、SPACE$、STRING$、LINE等を上手に使うと、画面が壊れたり、チラついたりせず、感じの良い、画面操作が行えます。要点をまとめると、下図のようになります。



(但し、xは0から255の数字であること。)

LOCATEで画面に書き込まれる場合と、TABで書き込まれる場合の違いは次のプログラムを実行し、対応するVRAM内容で確認して下さい。

```
          ABCDEF          GHIJKL          MNOPQR          STUVWX
Ok

list
10 LOCATE 0,0:PRINT TAB(10);"ABCDEF";SPC(10);"GHIJKL"
20 LOCATE 42,0:PRINT "MNOPQR";STRING$(10,32);"STUVWX"
Ok
mon
*DF300,F34F
F300 20 20 20 20 20 20 20 20 20 20 41 42 43 44 45 46
F310 20 20 20 20 20 20 20 20 20 20 47 48 49 4A 4B 4C
F320 00 00 00 00 00 00 00 00 00 00 4D 4E 4F 50 51 52
F330 20 20 20 20 20 20 20 20 20 20 53 54 55 56 57 58
F340 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
```

TABで表示位置を指定すると、PRINT文が実行される位置からTABで指定した位置まで、20で埋められます。(20は空白のCHR$(32)のアスキーコードです)しかしLOCATE文では直接位置を指定するため、00のままです。スクリーンコピーする場合、ダイレクトにPOKE&HEA58,1を実行し、ファンクションキー[f·9]キーを押せばコピーできます。00を20に変えておかなければ、画面上の空白は無視されてコピーされます。POKE &HEA5A, &H20 [RET] キーで背景は「00」から「20」に変ります。

文字数の異なる入力データをそろえて表示する1つの方法として、文字数の上限を設定しておき、入力文字が、上限に満たない部分はスペースで埋めて、","または";" で表示させます。

```
10 FOR I=0 TO 5
20 PRINT I;:INPUT A$(I)
30 IF LEN(A$(I))>10 THEN 20
40 NEXT I
50 FOR J=0 TO 5
60 PRINT A$(J);SPC(10-LEN(A$(J)));
70 NEXT J
```

このように、SPC, SPACE$, STRING$の(　)の中で変数、数式も使うことができます。

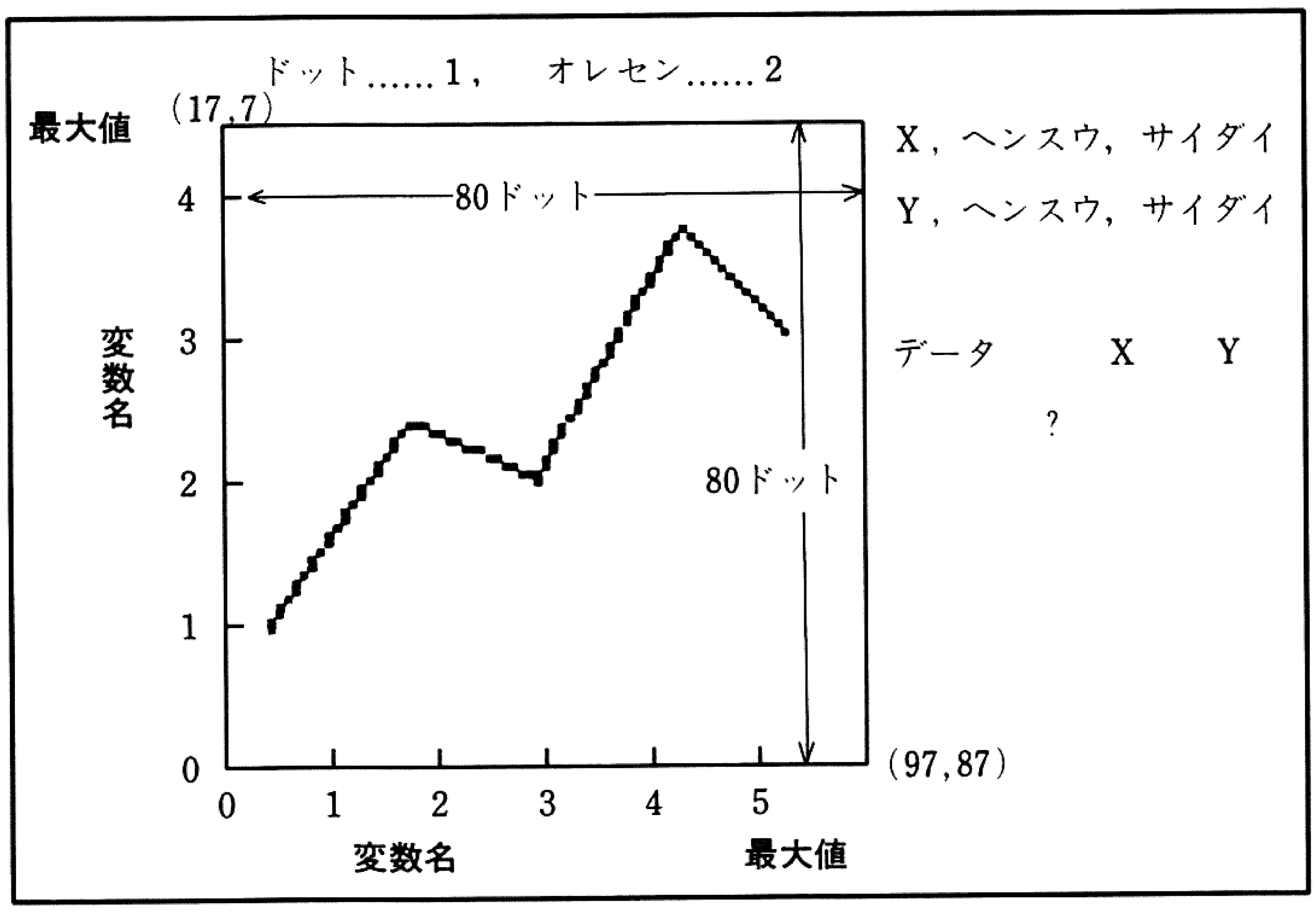
応用例として、次のプログラムを紹介します。消去のプログラムはサブルーチンとしておけば便利です。

```
10 LOCATE 10,10:PRINT "┌────────────────────────────┐
20 LOCATE 10,11:PRINT "│ ﾃﾞｰﾀ ｦ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ !!    │
30 LOCATE 10,12:PRINT "├──────────────┬─────────────┤
40 LOCATE 10,13:PRINT "│   TOKYO      │   OSAKA     │
50 LOCATE 10,14:PRINT "├──────────────┼─────────────┤
60 LOCATE 10,15:PRINT "│              │             │
70 LOCATE 10,16:PRINT "└──────────────┴─────────────┘
80 LOCATE 12,15:COLOR 2:PRINT "?":COLOR 0
90 LOCATE 12,18:INPUT A:GOSUB 170:LOCATE 12,15:PRINT A
100 LOCATE 28,15:COLOR 2:PRINT "?":COLOR 0
110 LOCATE 12,18:INPUT B:GOSUB 170:LOCATE 28,15:PRINT B
120 FOR I=0 TO 10:LINE(86+I,44+I)-(118-I,63-I),PSET,B
130 FOR J=0 TO 100:NEXT J :GOSUB 180:NEXT I
140 GOSUB 190:LOCATE 48,13:PRINT "END !!"
150 LOCATE 0,0 :END
160 '---------SUB ROUTINE--------------------------------------------
170 LOCATE 12,18:PRINT SPACE$(5):RETURN
180 LINE(43,11)-(59,15)," ",BF:RETURN
190 LOCATE 11,15:PRINT SPACE$(14):LOCATE 27,15:PRINT SPACE$(15):RETURN
```

2-7.グラフ作成

グラフ表示において考えねばならないことは、水平軸、垂直軸の単位の設定と目盛のとり方、それに、計算結果の表示位置の問題です。

〔図表12-16〕



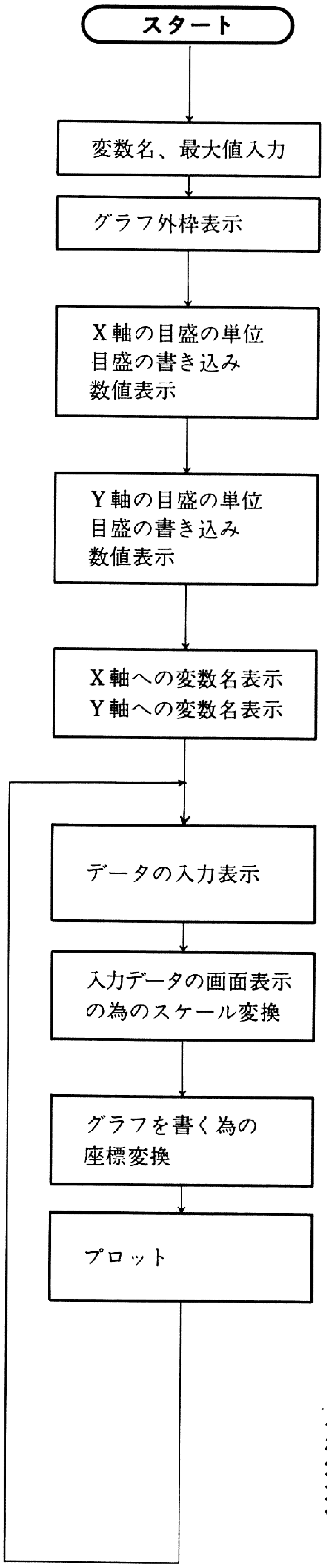
紹介するプログラムでは、変数名とその最大値を入力すると、自動的に目盛られ、変数名、最大値が表示されます。次に、データをその範囲内で入力しますと、ドット表示の場合は1ドットづつ表示され、折れ線の場合は、直前に入れたドットと次に入れたドットが直線で結ばれます。

データのプロットは、それぞれの軸に対して

$$\frac{\text{X 又は Y 軸の軸の長さ}}{(\text{データの最大値})-(\text{データの最小値})}\times(\text{データ})$$

により、グラフ上のポイントを出します。次に、スクリーン上の座標にセットするためにグラフの原点座標分だけ、増減すると、表示すべき位置にポイントをセットできます。

グラフ作成プログラムの流れは次の通りです。




```
10 CONSOLE 0,25,0,0:WIDTH 80,25:PRINT CHR$(12)
20 LOCATE 10,0:INPUT"ﾄﾞｯﾄ ........1   ,ｵﾚｾﾝ ...........2";W

30  LOCATE50,3:INPUT"X  ﾉ ﾍﾝｽｳ , ｻｲﾀﾞｲ";NA$(1),B1
40  LOCATE50,4:INPUT"Y  ﾉ ﾍﾝｽｳ , ｻｲﾀﾞｲ";NA$(2),B2

50 LINE(17,7)-(97,87),PSET,B

60 B=B1:GOSUB 310    '--------------------------------------X ﾉ ﾒﾓﾘ
70 I=0:FOR X2=17 TO 98 STEP SI
80 LINE(X2,86)-(X2,87),PSET
90 X4=X2/2-1:Y4=22:LOCATE X4,Y4:PRINT I;:I=I+1:NEXT X2

100 B=B2:GOSUB 310    '--------------------------------------Y ﾉ ﾒﾓﾘ
110 I=0:FOR Y2=87 TO 7 STEP -SI
120 LINE(17,Y2)-(18,Y2),PSET
130 Y4=Y2/4:X4=5:LOCATE X4,Y4:PRINT I;:I=I+1 :NEXT Y2

140 LOCATE 20,23:PRINT NA$(1);TAB(47);USING"###";B1 '------X ﾉ ﾍﾝｽｳ
150 LOCATE 1,1:PRINTUSING"###";B2                   '------Y ﾉ ﾍﾝｽｳ
160 FOR J=1 TO 10:A$=MID$(NA$(2),J,1)
170 LOCATE 4,4+J:PRINT A$:NEXT J

180 Z=0                  '--------------------------------ﾃﾞｰﾀ ﾉ ﾆｭｳﾘｮｸ
190 LOCATE50,10:PRINT"ﾃﾞｰﾀ    X    Y"
200 LOCATE55,11:INPUT X1:LOCATE60,11:INPUTY1
210 LOCATE55,11:PRINT SPACE$(15)
220 LOCATE55,12:PRINTUSING" ###    ###";X1;Y1

230 X1=X1*80/B1:Y1=Y1*80/B2

235 X1=X1+17:Y1=87-Y1

240 PSET(X1,Y1)          '-------------------------------------- ﾌﾟﾛｯﾄ
250 IF Z<>0 THEN LINE(X2,Y2)-(X1,Y1),PSET

260 Z=Z+1
270 IF Z>4 THEN LOCATE 50,20:END
280 X2=X1:Y2=Y1
290 IF W=1 THEN Z=0
300 GOTO 190
310 C$=STR$(B):C$=LEFT$(C$,3):C=VAL(C$):SI=800/C:RETURN
```

上のプログラム例では、X軸、Y軸の変数の値を指定しましたが、Xを変数としたYの関数を、190行～220行に導入することにより、関数プロットが出来ます、上の例では、ドット、折れ線グラフでしたが、棒グラフへの表示変更も簡単にできます。

データマトリックス(2次元配列)へのデータ入力

汎用性のある2次元配列へのデータ入力プログラムを紹介しましょう。任意の行、列、データの最大文字数を入力することにより、行列表が表示されます。

書き込む位置が一目でわかるよう?マークをその位置でブリンクさせ入力時のエラーにより画面が壊れるのを防ぐ為に下にスクロールウィンドウを設けました。

```
         1       2       3       4       5

 1     286      17     342     279     371
 2      35      35       3      43      55
 3       6     477      34     684      58
 4     618     279     394     473     242
 5     625     351     256     408     265
 6     288     312     336     360     384



ﾃﾞｰﾀ ﾉ ﾃｲｾｲ ｼﾏｽｶ (y/n) ? y
ﾃｲｾｲ ｽﾙ 「ｷﾞｮｳ」ﾄ「ﾚﾂ」ｦ ﾆｭｳﾘｮｸ
?
```

スクロールウィンドウ(上の画面の最下部2行)

配列の添字と、LOCATE文の列と行番号を対応させることにより入力、訂正の位置指定が楽になります。

```
10 INPUT"ﾚﾂ";M:INPUT"ﾓｼﾞｽｳ";L:X=M*(L+2)+5 :PRINT X:P$=STRING$(L,35)
20 IF X>79 THEN PRINT "ﾓｼﾞｽｳ (L)  ｦ ﾁｲｻｸ ｽﾙｶ,ﾚﾂ(M) ｦ ﾁｲｻｸ ｼﾅｻｲ":GOTO 10
30 INPUT"ｷﾞｮｳ";N:IF N+5>17 THEN PRINT "ｷﾞｮｳ (N) ｦ ﾁｲｻｸ ｼﾅｻｲ":GOTO 30
50 PRINT CHR$(12):GOSUB 400:CONSOLE 20,4
60 FOR I=1 TO N:FOR J=1 TO M:GOSUB300:GOSUB200:NEXTJ,I:GOSUB320
100 CONSOLE 0,25:END
200 COLOR2:LOCATEXI,YJ:PRINT "?":COLOR0                    '----------sub 1
210 LOCATE 0,20:PRINT SPACE$(255)
220 LOCATE 0,20:INPUT"ﾃﾞｰﾀ";A#(I,J)
230 LOCATEXI,YJ:PRINT SPACE$(L+1)
240 LOCATEXI+1,YJ:PRINTUSING P$;A#(I,J):RETURN
300 XI=(J-1)*(L+2)+5:YJ=I+5:RETURN                         '----------sub 2
320 Q$=" "                                                 '----------sub 3
325 LOCATE 0,21:PRINT "ﾃﾞｰﾀ ﾉ ﾃｲｾｲ ｼﾏｽｶ (y/n)";Q$;:INPUTQ$
330 IF Q$="n"THEN RETURN
340 PRINT "ﾃｲｾｲ ｽﾙ 「ｷﾞｮｳ」ﾄ「ﾚﾂ」ｦ ﾆｭｳﾘｮｸ"                    '----------sub 4
350 INPUTI,J:GOSUB 300:GOSUB200:GOTO320
400  FOR I=0 TO M-1 :LOCATE (I*(L+2)+4+L),4:PRINT I+1:NEXT
410  FOR I=1 TO N  :LOCATE 2,I+5:PRINT USING"##";I:NEXT :RETURN
```

第3章　キー入力

第3章　キー入力

3-1.ファンクションキー

3-1-1 メモリー状の格納状態

KEYLIST でファンクションキーの内容を表示してみると初期設定では、次のようになってます。

```
[illegible]          time$
auto                 key
go to                print
list                 list.↵↑↑
run↵                 cont↵
```

これらはメモリー上には、EA7C番地からEB1B番地に格納されています。その状態をモニターモードで調べることができます。

メモリの内容の20～F7
をキャラクター表示したもの

```
mon
*DEA70,EB1F
EA70  00 00 00 00 00 00 00 00  00 01 01 79 09 00 00 00  ........ ...y....
EA80  00 00 00 00 00 00 00 00  00 00 00 00 61 75 74 6F  ........ ....auto
EA90  20 00 00 00 00 00 00 00  00 00 00 00 67 6F 20 74   ....... ....go t
EAA0  6F 20 00 00 00 00 00 00  00 00 00 00 6C 69 73 74  o ...... ....list
EAB0  20 00 00 00 00 00 00 00  00 00 00 00 72 75 6E 0D   ....... ....run.
EAC0  00 00 00 00 00 00 00 00  00 00 00 00 74 69 6D 65  ........ ....time
EAD0  24 00 00 00 00 00 00 00  00 00 00 00 6B 65 79 20  $....... ....key
EAE0  00 00 00 00 00 00 00 00  00 00 00 00 70 72 69 6E  ........ ....prin
EAF0  74 20 00 00 00 00 00 00  00 00 00 00 6C 69 73 74  t ...... ....list
EB00  2E 0D 1E 1E 00 00 00 00  00 00 00 00 63 6F 6E 74  ........ ....cont
EB10  0D 00 00 00 00 00 00 00  00 00 00 00 00 2F B7 C9  ........ ....・/ｷﾉ
*
```

ファンクションキーの境界のための「00」

1つのファンクションキーには16 Byte使用されていますが、最後の1 Byteは次のファンクションキーの内容と区別する為に、自動的に「00」が書き込まれます。ユーザーは、15 Byte、すなわち15文字しか定義できなく、それ以上は無視されます。

3-1-2 ROM内の格納状態

N-BASICのROM内の3A81番地から3B20番地までにも、前頁と同じ内容が格納されていますので、モニターモードで表示してみましょう。

```
mon
*D3A81.3B20
3A81 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3A90 00 61 75 74 6F 20 00 00 00 00 00 00 00 00 00 00
3AA0 00 67 6F 20 74 6F 20 00 00 00 00 00 00 00 00 00
3AB0 00 6C 69 73 74 20 00 00 00 00 00 00 00 00 00 00
3AC0 00 72 75 6E 0D 00 00 00 00 00 00 00 00 00 00 00
3AD0 00 74 69 6D 65 24 00 00 00 00 00 00 00 00 00 00
3AE0 00 6B 65 79 20 00 00 00 00 00 00 00 00 00 00 00
3AF0 00 70 72 69 6E 74 20 00 00 00 00 00 00 00 00 00
3B00 00 6C 69 73 74 2E 0D 1E 1E 00 00 00 00 00 00 00
3B10 00 63 6F 6E 74 0D 00 00 00 00 00 00 00 00 00 00
3B20 00
*
```

〔図表3-1〕

	RAM	ROM
F1	EA7C	3A81
2	EA8C	3A91
3	EA9C	3AA1
4	EAAC	3AB1
5	EABC	3AC1
6	EACC	3AD1
7	EADC	3AE1
8	EAEC	3AF1
9	EAFC	3B01
10	EB0C	3B11

これはPCをスイッチオンした時、ファンクションキーのイニシャライズに使われています。これを用いて、プログラム中あるいは、ダイレクトにKEYコマンドを使って、イニシャライズするよりも、次に示す、ステートメントを実行する方が、はるかにスマートな方法です。なお、このステートメントの実行後、[SHIFT]キーを押すか、またはWIDTHを実行するとイニシャライズされたキーの内容が表示されます。

```
FOR I=0 TO 159:POKE&HEA7C+I,PEEK(&H3A81+I):NEXT I
```

3-1-3 内容の定義の仕方

ファンクションキーは、" " で囲まれた文字列とコントロールコードを「＋」する形で、定義することができ、一種のプログラミングができます。15文字内で定義できないときには、境界を示す「00」にPOKE文でキャラクタを書き込んでやることにより、隣のファンクションキー領域も使用できます。

```
10 A$=CHR$(13):B$=CHR$(28):C$=CHR$(30):D$=CHR$(34)
20 PRINT "F-1 Key ヲ オシナサイ !!"
30 KEY1,"print"+D$+"load"+D$+A$+C$+C$+B$
40 KEY2,B$+B$+B$+D$+"file"+A$
50 POKE &HEA8B,28
```

このサンプルプログラムは、自動的に他のプログラムをロードする例で、40行のfileの個所を、ファイル名になおしますと、望むプログラムをオートロードできます。

なお、CHR$(28)、CHR$(30)はカーソル移動のコントロールコードです。モニターモードで、f·1、f·2の内容を調べてみましょう。

```
on
%DEA7C,EA9B
EA7C 70 72 69 6E                                  EA8B
EA80 74 22 6C 6F 61 64 22 0D 1E 1E 1C [1C] 1C 1C 1C 22
EA90 66 69 6C 65 0D 00 00 00 00 00 00 00
%
```

境界を示す「00」が1C〔CHR$(28)〕と書き換えられ、f·1とf·2がリンクされています。これでf·1を押すだけで、f·1とf·2の内容を実行(表示)することができます。

3-1-4 キーポインタとファンクションキーフラグの使い方

今までは、ファンクションキーを必ず押さなければ、キーの内容が実行されませんでしたが、ポインタとフラグを使うことにより、自動的にキーが押されたと同じ動作をさせることができます。次は自動的にLISTするプログラム例です。

```
list
10 KEY1,"list"+CHR$(13)
20 POKE&HEDC0,&H7C:POKE&HEDC1,&HEA
30 POKE&HEA68,1
Ok


run
Ok
list
10 KEY1,"list"+CHR$(13)
20 POKE&HEDC0,&H7C:POKE&HEDC1,&HEA
30 POKE&HEA68,1
Ok
```

キーポインタはEDC0とEDC1、キーフラグはEA68で、初期値は、「00」です。

N-BASICはファンクションキーが押されると、どのキーが押されたかを判断し、キーポインタの値をEDC0(下位)、EDC1(上位)に入れます。

それで、キーポインタに、使用するファンクションキーの先頭番地を入れてやり、キーフラグに1を立ててやると、プログラム実行後、ファンクションキーが押されたものとみなされ自動的にその内容が実行されます。

〔図表3-2〕

	ファンクションキーポインタ	
	EDC0	EDC1
F. Key 1	7C	EA
2	8C	EA
3	9C	EA
4	AC	EA
5	BC	EA
6	CC	EA
7	DC	EA
8	EC	EA
9	FC	EA
10	0C	EB

3-1-5 長いメッセージの場合

ファンクションキー領域を使わないで、プログラムのフリーエリアにメッセージを格納し、ファンクションキーポインタにその先頭番地を入れ、キーフラグを立ててやることにより、254文字以下の長いメッセージを出力表示することができます。

```
10  INPUT A$:Q=LEN(A$):FOR S=1 TO Q
20  T$=MID$(A$,S,1)
30  POKE &HA000+S-1,ASC(T$):NEXT
40  POKE &HA000+Q,0
50  POKE &HEDC0,&H0:POKE &HEDC1,&HA0
60  POKE &HEA68,1
```

また、複数個のコマンドを文字列として設定しておき、VARPTRで、そのアドレスを調べキーポインタにPOKEすることにより、自動的に1度に実行させることもできます。

```
100 A$="mount1:files:run"+CHR$(34)+"TEST"+CHR$(34)+CHR$(13)+CHR$(0)
110 POKE&HEDC0,PEEK(VARPTR(A$)+1):POKE&HEDC1,PEEK(VARPTR(A$)+2)
120 POKE&HEA68,1
```

（注意）境界を示す「00」の前に、「00」があると、読み込み終了となりますので、途中の余分な「00」はスペースコード&H20(32)に書き換えて下さい。

3-1-6 DATA文の自動作成

メモリー上のある任意の番地の機械語の内容をBASICプログラムのDATA文に自動的につくり変えます。

```
10  A=8020:L=1000
20  FOR I=100 TO 105:PRINT L+I;"data";
30  FOR J=0 TO 15:P=PEEK(A):N$=RIGHT$("00"+HEX$(P),2)
40  A=A+1:PRINT N$;",";:NEXT J:PRINT CHR$(29);" ":NEXT I
50  KEY6,STRING$(7,30)+STRING$(6,13)
60  POKE&HEDC0,&HCC:POKE&HEDC1,&HEA:POKE&HEA68,1
```

STRING$ (7,30)はカーソル上移動CHR$(30)を7回実行しSTRING$(6,13)でCHR$(13)を6回実行させ、BASICプログラム上につくりかえます。

3-1-7 メモリー領域への直接書き込み

```
10 KEY6,"mon"+CHR$(13)+"D8020,805f"+CHR$(13):POKE&HEADB,32
20 KEY7,"sf300"+CHR$(13)+"00"+CHR$(13)+CHR$(2)
30 POKE&HEDC0,&HCC:POKE&HEDC1,&HEA
40 POKE&HEA68,1
```

モニターモードにして、8020番地から805F番地までを表示し、VRAMへの書き込みした後、BACICへ戻ってくる例です。

3-1-8 DATA文からの書き込み

KEYコマンドには変数を使うことが可能です。ビジネスユースの場合、所定の文字列を使うことが多いので、文字列群をDATA文に格納しておき、場合に応じて、ファンクションキーに定義してやることにより、キー入力が非常に簡単に行えます。

```
5 CONSOLE 0,25,1,0
10 DATA ﾄｳｷｮｳ,ｵｵｻｶ,ﾅｺﾞﾔ,ｻｯﾎﾟﾛ,ﾌｸｵｶ
20 DATA ﾆｭｳﾖｰｸ,ﾗｽﾍﾞｶﾞｽ,ｼｶｺﾞ,ｻﾝﾌﾗﾝｼｽｺ,ﾜｼﾝﾄﾝ
30 PRINT "ｸﾆ ｦ ｴﾗﾝ ﾃﾞ ｸﾀﾞｻｲ  ﾆﾎﾝ(j)/ｱﾒﾘｶ(a)/ｵﾜﾘ(ｘ)":INPUT A$
40 IF A$="j" THEN RESTORE 10:FOR I=1 TO 5:READ B$(I):NEXT I:GOTO 60
50 IF A$="a" THEN RESTORE 20:FOR I=1 TO 5:READ B$(I):NEXT I:GOTO 60
55 IF A$="ｘ" THEN 100
58 GOTO 30
60 FOR  I=1 TO 7:KEY I,B$(I):NEXT I
70 INPUT "ﾄｼ ﾒｲ";C$:PRINT "<< ";C$;" >>"
80 GOTO 30
100 FOR I=0 TO 159:POKE&HEA7C+I,PEEK(&H3A81+I):NEXT
110 PRINT CHR$(12):END
```

60行でDATA文から文字列を読み込んだ配列変数をそれぞれのファンクションキーに割り当てます。100行はファンクションキーの初期化です。

3-1-9 キー選択

ビジネスユースの場合、いくつかのメニューの中から、特定の仕事を選び、サブルーチンへ飛ぶことがよくあります。そのような場合のファンクションキーの使い方をあげておきます。

```
5 CONSOLE 0,25,1,0:PRINT CHR$(12)
10 DATAﾄｳﾛｸ ,ｴﾂﾗﾝ ,ﾌﾟﾘﾝﾄ,ﾃｲｾｲ ,ｻｸｼﾞｮ
20 FOR I=1 TO 5:READ B$(I)
30 LOCATE 25,5+I*2:PRINT B$(I);".....F-";I
40 KEYI,B$(I)
50 NEXT I
60 A=255-INP(9)
70 FOR N=1 TO 5
80 IF A=2^N THEN 100 ELSE NEXT N
90 GOTO 60
100 PRINT CHR$(12):ON N GOSUB 200,300,400,500,600
110 FOR J=0 TO 159:POKE&HEA7C+J,PEEK(&H3A81+J):NEXT J:END
200 PRINT B$(1):RETURN
300 PRINT B$(2):RETURN
400 PRINT B$(3):RETURN
500 PRINT B$(4):RETURN
600 PRINT B$(5):RETURN
```

60行のINP(9)がファンクションキーのI/Oポートのデータです。(詳しくはINP文を参照下さい)

3-2.キー入力ステートメント

3-2-1 INPUT

キー入力比較表(後出)の他に補足すると、

- 入力した場所……？マークが残る。
- BREAKした時…Break in行の表示があり、3行下にカーソルが移動する。
- 入力誤りの時……？Redo from startの表示があり、2行下に再入力待ちとなる。
- 画面の下で入力した時…………画面がスクロールされて最上位の行は消えてしまう。

INPUT文を表形式の中で使う場合、これらの対策を考えてないと表が崩れたり、次の入力の時、非常にやり難く、最初から入力してやらねばならなくなります。対策として、LOCATE文で入力データの表示位置を決めておくか、console文で、処理してやります。次に例を示します。

```
10 CONSOLE 20,5,0,0:PRINT CHR$(12)
20 FOR I=0 TO 9
30 LOCATE 0+S,20+I:PRINT I;:INPUT")...";A(I)
40 LOCATE 0,0+I:PRINT SPACE$(20)
50 LOCATE 0,0+I:PRINT I;")";A(I)
60 NEXT I
70 LOCATE 0,24:PRINT "ﾃｲｾｲ ｼﾏｽｶ (y/n)"
80 X$=INKEY$:IF X$="y" THEN S=S+20:GOTO 20
90 IF X$<>"n" THEN 80
100 END
```

30行のSは再入力時、以前入力したデータに重ならないようにしたものです。前に入力したデータを修正する場合、配列の形にしておくと、修正しなくてもよいものは、RETキーを押すだけで済みますが、単純変数にしておくと、最後に入力したデータの内容が入力されます。

また入力時の？マークを消すためには次の方式があります。

```
10 PRINT CHR$(12)
20 FOR K=0 TO 5
30 LOCATE 1,1+K:PRINT USING"###";K
40 LOCATE 5,1+K:INPUT A(K)
50 LOCATE 5,1+K:PRINT USING"#####.##";A(K)
60 NEXT K
```

3-2-2 INPUT$( )

（　）の中の数値によって入力文字数を指定することができます。このときカーソルは入力位置にあり、入力中は移動しませんし、エコーバックもありません。そのかわりカーソル移動キーの入力が可能ですので、複雑なカーソル移動が容易にできます。

```
10 A$=INPUT$(10)
20 FOR I=0 TO 20:FOR J=1 TO 10
30 LOCATE 10+I,8+J:PRINT A$
40 NEXT J,I
```

上のプログラムをRUNさせて、

[↓][↓][K][→][→][↓][L][→][↓][P]と入力して下さい。

また、入力文字数を指定できるので、キーワードによる文字列の検索に使うと便利です。

```
10 PRINT "ｷｰﾜｰﾄﾞ ｦ ｲﾚﾃ ｸﾀﾞｻｲ  ";
20 A$=INPUT$(1):X=INSTR("abcdefghij",A$):IF X=0 THEN 10
30 PRINT X,A$
```

3-2-3 LINEINPUT

LINEINPUT A$とINPUT A$は殆んど同じですが、大きな違いは、LINEINPUTの場合、“,”が入力できること、入力文字無しでRETキーを押した場合、ヌルストリングが返ってくること、それに、？マークの表示が無いことです。数多くのデータを入力し、後で修正する必要が発生し易い場合は、INPUT文を使った方が良いでしょう。長いメッセージ文で“,”が入るものは、？マークの残らない、LINEINPUT文を使用した方が良いと思います。

3-2-4 INKEY$

このステートメントの特徴は、1文字入力でしかも、キー入力を待たないで、次の処理に進むことです。ですから、ループを作ってやって、INKEY$で判断させ、文字に対応した行へ飛ばせる使い方が一般的です。

```
10 Y$=INKEY$
20 IF Y$=CHR$(13) THEN 30 ELSE IF Y$="a" THEN 40 ELSE 10
30 PRINT "<< END >>":END
40 PRINT " O K  !! "
```

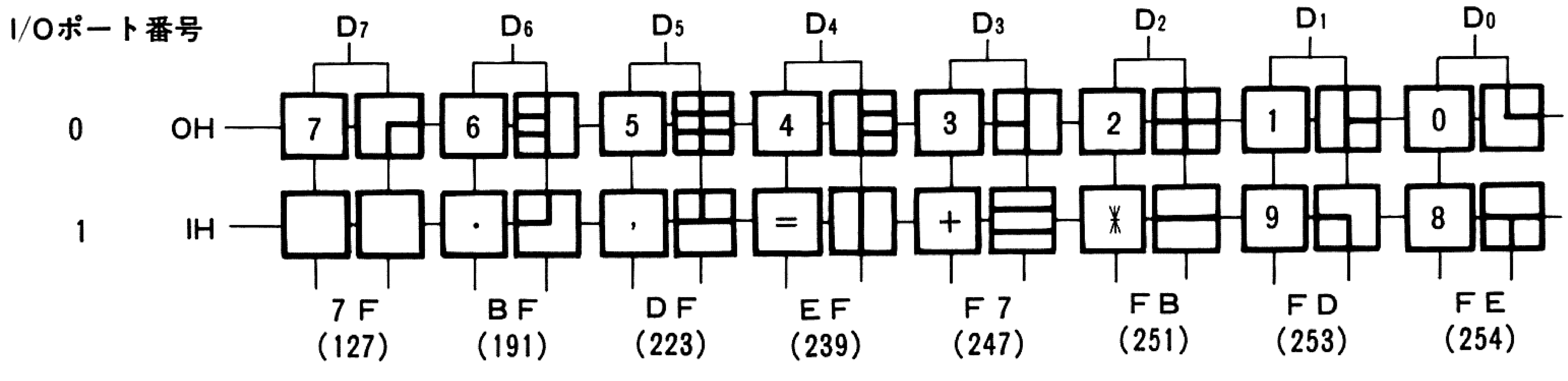
キーが押されなければ、入力待ちで、キー入力により、その文字に対応した処理を行ないます。また10行に戻すことにより、常にキー入力チェックを行ってます。INKEY$の場合はコントロールキーも入力できます。

3-2-5 INP

I/Oポート0～9を指定することにより、キーボードからの、8ビットのデータを入力することができます。キーが押されると、そのキーに対応するビットが0になるので、ポートのデータで、どのキーとどのキーが押されたか判別することができます。次にポート0、ポート1のキーマトリックスと、それぞれのキーが押された時のデータの内容を示します。

（詳しくは、I/Oポート表を参照下さい。）

〔図表3-3〕



5のキーが押されたときの、ポート0のデータは2進表示で11011111、16進数表示ではDFH、10進数では223となります。同時に5と2のキーが押された時は11011011となりDBH(219)を与えます。このデータの内容を判別するプログラムにより、押されたキーに対応する処理をさせることができます。次にいくつかの例をあげてみましょう。

（I）
```
10 A=INP(0)
20 IF A=251 THEN PRINT "key..2"
30 IF A=223 THEN PRINT "key..5"
40 IF A=219 THEN PRINT "key..2 and key..5"
50 GOTO 10
```

どのキーが押されたかを示す簡単なプログラムで、GOTO10により常にキー入力を問い合せキー入力がなかったら、次の処理に移ってゆきます。ここでINP(0)＝253 の形を使う方法もあります。

（II）
```
10 WIDTH 80,25:PRINT CHR$(12)
20 X=80:Y=50
30 A=( NOT INP(0) AND 255)
40 B=( NOT INP(1) AND 255)
50 IF A AND 2  THEN X=X-1
60 IF A AND 8  THEN X=X+1
70 IF A AND 32 THEN Y=Y-1
80 IF B AND 32 THEN Y=Y+1
90  IF Y<0 OR Y>95  THEN Y=(Y +96)MOD 96
100 IF X<0 OR X>159 THEN X=(X+160)MOD 160
110 IF POINT(X,Y)=-1 THEN 120 ELSE PSET(X,Y)
120 GOTO 30
```

このプログラムは、ドットパターンで自由にCRT上に図形を描かせるプログラムです。ここではNOT、ANDなどの論理演算子を使って判断させています。操作するキーはテンキーボードの[5](上)、[・](下)、[1](左)、[3](右)で、2つのキー(たとえば1と5)を同時に押せば、ななめにも動きます。

（III）テンキーをHEX KEYに

```
10 '------------------Set Memory---------------------------
20 G$=CHR$(223)+CHR$(191)+CHR$(127)+CHR$(239)+CHR$(247)+CHR$(251)
30 H$="ABCDEF"
40 INPUT"Address";MA$:MA$="&H"+MA$:AD=VAL(MA$)
50 PRINT RIGHT$("000"+HEX$(AD),4)"  ";
60 FOR J=1 TO 16:FOR K=1 TO 2:M$=""
70 A$=INPUT$(1):IF A$="" THEN 70
80 IF A$>="0"AND A$<="9" THEN M$=A$:GOTO 100
90 N=INSTR(G$,CHR$(INP(1))):IF N=0 THEN 70 ELSE M$=MID$(H$,N,1)
100 M1$=M1$+M$:PRINT M$;:GOSUB 160
110 NEXT K:PRINT "  ";
120 F=VAL("&H"+M1$):POKE AD,F:AD=AD+1:M1$=""
130 NEXT J:PRINT
140 V$=RIGHT$(HEX$(AD),1):IF V$<>"8" AND V$<>"0"THEN 60
150 GOTO 50
160 FOR I=0 TO 100:NEXT :BEEP 1:BEEP 0:RETURN
```

3-2-6 WAIT

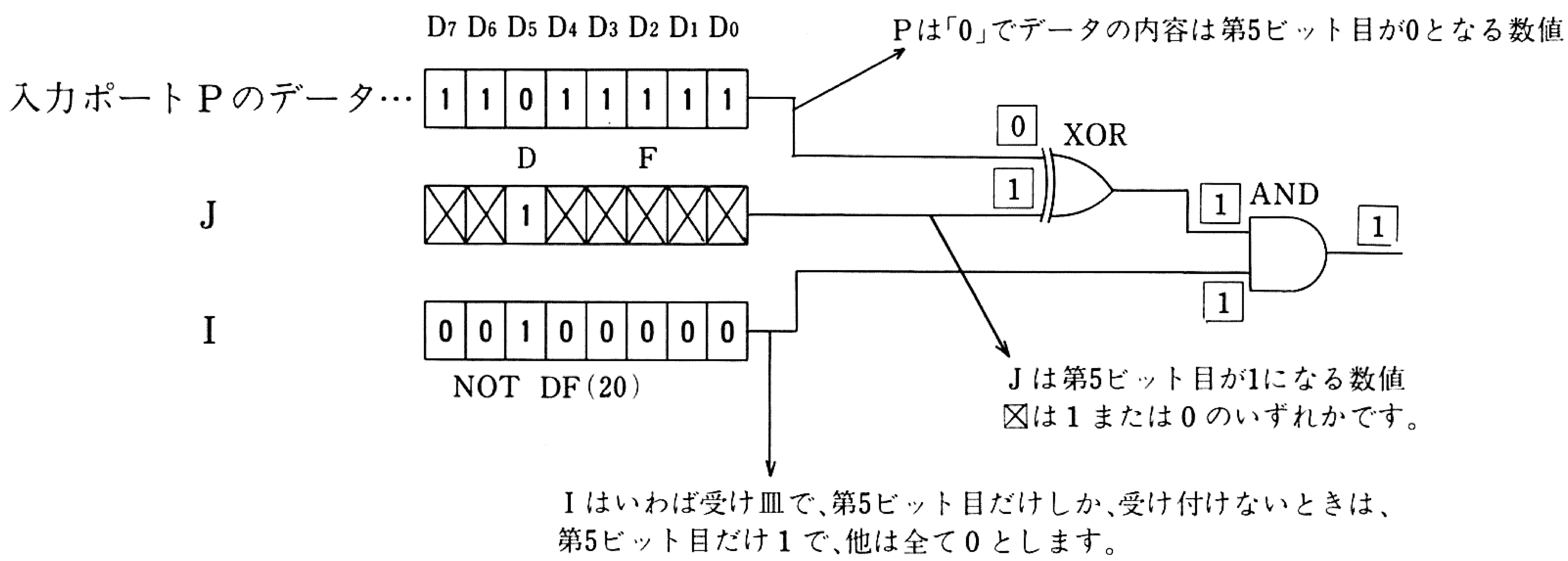
WAIT文は、WAIT　P,I,Jの書式で使われます。

入力ポートPから入力されるデータで指定されたビットの内容と、Jで与えられた数値で指定されるビットの内容の排他的論理和(XOR)がIの数値で指定されるビットの内容とのANDで、1(真)になることにより、次の行へ進みます。

その間プログラムの実行は停止します。(入力ポートを指定するときは、I/0ポート一覧表を参照して下さい)

テンキーの「5」が押されるまでWAITする場合のI,Jの数値の決め方の手順は、

1. キーに対応した、ポート番号とデータ内容を決める──→0ポート,DF
2. キーに対応したビットが1になるようにIを決める──→NOT DF(20)
3. Iで指定したビットが「1」となるJの値を決める　──→2F又はFF,その他



WAIT 0,&H20,&H2F　　　　　　又は WAIT 0,32,47
　　　　(&HFF その他)　　　　　　　　　(　255　その他)

のようになります。

ここで応用例を紹介しましょう。

他の人に自分のプログラムを使われたくない場合、パスワードを設定し、LISTして、解読しようとしても、その行だけLISTされないプログラムです。。まず次のプログラムをキーインして下さい。

```
10 PRINT CHR$(12)
20 PRINT"*******************************************************
********************************* PASS WORD ヲ KEY IN シテクダサイ ******************
************************************************************
*********     "
25 WAIT4,&H8,&HF8:WAIT3,&H80,&H8F:WAIT2,&H40,&H4F:WAIT4,&H10,&H1F
30 PRINT CHR$(12):LINE(0,0)-(159,89),PSET,B
40 LOCATE 37,11:PRINT "OK !!"
50 GOTO 40
```

RUNさせて下さい。パスワードは「soft」です。

次にモニターモードで8020番地から819F　番地までの内容を表示し、リンクポインタをチェックして下さい。(92ページへ続く)

3-2-7 キー入力比較表

		INPUT	LINEINPUT	INPUT$	INKEY$
	書式	INPUT A / A$	LINE INPUT A$	A$=INPUT$(X)	A$=INKEY$
入力表示	プロンプト〔″ ″〕	可能	可能	できない	できない
入力表示	入力指示の？表示	有	無	無	無
入力表示	カーソル表示	有	有	有	無
入力表示	入力時のカーソル移動	動	動	不動	無
入力表示	エコーバック	有	有	無	無
入力表示	入力待ち	待つ	待つ	待つ	待たない
キー入力	入力文字の種類：英、数、片仮名 ファンクション グラフィック	A：数字のみ A$：可能	可能	可能	可能
キー入力	入力文字の種類：特殊文字	カンマは変数の区切り	可能	可能	可能
キー入力	入力文字の種類：コントロールコードの入力	できない	できない	可能	可能
キー入力	入力文字数	A：39桁以内[1)] A$：254文字以内	254文字以内	指定した文字数（254文字以内）	1文字
キー入力	実行	[RET]キー	[RET]キー	自動的	自動的
キー入力	入力文字なしで[RET]キー	前に入力した内容で再入力	ヌルストリング（空白）	CHR$(13)が入力される	CHR$(13)が入力される
入力中断	[STOP]キー	中断	中断	中断	中断
入力中断	[CTRL] [C]	中断	中断	中断	中断しない
入力中断	BREAK時のBEEP音	音が出る	音が出る	音は出ない	音が出る
入力中断	CONTによるプログラム再開	可能	可能	不可能	可能

注）1）変数の型を宣言することにより、入力数値の有効桁数が決まります。

3-2-8 キー入力ステートメントの使い方

	メ リ ッ ト	注 意 点	使 い 方
INPUT	○プロンプト文によりメッセージを使用でき、入力を促す"?"マークが表示される。 ○配列の形にしておくと、データ修正時、変更しない場合は[RET]キーのみ押せばよい。	○実行後入力位置より3行下にカーソルが移動するため最上位行がスクロールされる。 ○入力誤りの時、画面が壊れる。 ○単純変数の場合、データ入力せず、RETキーのみ押すと最後に入力したデータがそのまま入力される。	○25行モードの場合22行目以内でのキー入力をする。 ○スクロールウィンドウを設ける。 ○LOCATE文でカーソル移動する。 ○後で修正が必要なデータの入力に使うと便利。
LINE INPUT	○プロンプト文によりメッセージが使用でき、入力を指示できる。 ○?マークが表示されないので文章を入力する場合の不自然さを感じさせない。 ○,(コンマ)も含めて入力できる。	○画面の80桁目の位置の文字の欠落に注意。(N-BASIC Ver1.0の場合)。	○文章の入力に適している。
INPUT$	○必要な文字数を指定でき、入力後自動的に次のステートメントに進む。 ○コントロールキーの入力が可能。	○入力したものが画面に表示されない。	○キーワードによる文字列の検索に便利。
INKEY$	○1文字入力で自動的に次に進む。 ○コントロールコードが入力できる。	○同　上	○次の画面表示の待機に利用。 ○YES/NOの判断に便利。

アドレス	リンクポインタ
8021	2E80
802E	2A81
812A	5A81
815A	7A81
⋮	⋮

リンクポインタは左のようになっていますので25行目のWAIT文を表示しないようにするため、802E番地の「2A81」を次の行である「5A81」に、Sコマンドで、直接書き直して下さい。LISTしてみると、25行はリストされませんが、RUNさせると、同じように実行されます。

3-3.コントロールキー

3-3-1 コントロールコード

キャラクタコード表において、00H～1FHの部分は、コントロールキャラクタになっており、画面上やプリンタに出力すると、特別の動作をするものもあります。

例えば、コード表において、0CH(シンボルC_L、10進数で12)のコードは、キャラクタとして画面にプリントすると、画面を消去するという機能をもっています。

print chr$(&HC) または、print chr$(12)

また、同じコードをプリンタに出力すると、用紙を次の頁の先頭まで送るという機能になります。

lprint chr$(&HC) または、lprint chr$(12)

画面上にprint文で出力したときとくに機能のないものについては、対応するシンボルが表示されます。

```
print chr$(&H1B)
EC
OK
```

機能があるものについてはpoke文で直接VRAM(F300~FEB7)に書き込めばそのシンボルを表示することができます。次のプログラムで試してみて下さい。

```
S S E E E A B B H L H C C S S D D D D D N S E C E S E  → ← ↑ ↓
 H X X T Q K L S T F M L R O I E 1 2 3 4 K N B N M B C
list
100 FOR I=0 TO &H1F:POKE&HF300+I*2,I:NEXT
OK
```

もちろん、全部のキャラクタコードをVRAMにpoke文で書込んで表示することもできます。

```
S S E E E A B B H L H C C S S D D D D D N S E C E S E  → ← ↑ ↓
 H X X T Q K L S T F M L R O I E 1 2 3 4 K N B N M B C
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ ¥ ] ^ _
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▏ ▎ ▍ ▌ ▋ ▊ ▉ ┼ ┴ ┬ ┤ ├ ▔ ─ │ ▕ ┌ ┐ └ ┘ ╭ ╮ ╰ ╯
  。 「 」 、 ・ ヲ ァ ィ ゥ ェ ォ ャ ュ ョ ッ ー ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ
タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ ヘ ホ マ ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ン ゛ ゜
= ╞ ╪ ╡ ◢ ◣ ◥ ◤ ♠ ♥ ♦ ♣ ● ○ ╱ ╲ ╳ 円 年 月 日 時 分 秒
list
200 FORI=0TO&HFF
210 POKE&HF300+(I¥32)*120+(IMOD32)*2,I
220 NEXT
OK
run
OK
```

これらのコントロールコードに対応するキーには、次のものがあります。(　)内は10進。

〔図表3-3〕

08(8)…	INS DEL	1文字削除	12(18)… SHIFT + INS DEL　1文字あける
1B(27)…	ESC	実行の1時中断	
1C(28)…	⇆	カーソルを1つ右へ	1D(29)… SHIFT + ⇆　カーソルを1つ左へ
1E(30)…	⇅	カーソルを上の行へ	1F(31)… SHIFT + ⇅　カーソルを下の行へ

プログラム上での使い方はPRINT CHR$(&H0C)またはPRINT CHR$(12)の書式で使います。

また、すべてのコントロールコードは、CTRL キーとアルファベット大文字キーの組合わせで入力することができます。

例えば、CTRL + A で01H、CTRL + G で07Hとなりブザーが鳴るなどです。

すなわち、CTRL キーを押すことにより、入力キーの第6ビットを0にしていることを意味します。

例.

	7	6	5	4	3	2	1	0	
"R"…52H	0	1	0	1	0	0	1	0	
		↓							
CTRL +"R"…12H	0	0	0	1	0	0	1	0	……"D_2"

次のプログラムを実行すると、キー入力したキャラクタコードを10進数と16進数に変換して表示しますので、ESC キー、カーソルキー、INS/DEC キー、CTRL キーと@～Zの組合わせなどを入力して確認してください。

```
100 K$=INPUT$(1)
105 POKE&HF300+120*CSRLIN,ASC(K$)
110 LOCATE4,CSRLIN:PRINTUSING"&& ###";HEX$(ASC(K$)),ASC(K$)
120 GOTO100
```

〔図表3- 〕　　　　　　　　　　**コントロールコード表**

16進	10進	シンボル	シンボルの意味	対応するキー	機能
00	0		null	[CTRL] +@	スペース
01	1	SH	Start of Heading(ヘッディング開始)	〃 +A	
02	2	SX	Start of Text(テキスト開始)	〃 +B	カーソルを1項目ごとに左へ移す。
03	3	EX	End of Text(テキスト終了)	〃 +C	STOPと同意(プログラムの実行を中止して、N-BASICコマンドモードに戻る
04	4	ET	End of Transmission(伝送終了)	〃 +D	
05	5	EQ	Enquiry(問合わせ)	〃 +E	カーソル位置から後ろをまっ消する。
06	6	AK	Acknowledge(肯定応答)	〃 +F	
07	7	BL	Bell(ベル、ブザー)	〃 +G	内蔵のブザーを0.5秒間鳴らす。
08	8	BS	Back Space(後退)	〃 +H	[DEL] カーソルのすぐ左の1文字を削除する。
09	9	HT	Harizontal Tabulation(水平タブ)	〃 +I	8文字毎の水平タブ
0A	10	LF	Line Feed(改行)	〃 +J	行を2つに分ける。 カーソルを次の行へ移す。
0B	11	HM	Home (VT) Vertical Tabulation(垂直タブ)	〃 +K	カーソルをホームポジション画面左上に戻す。
0C	12	CL	Clear (FF) Form Feed(改頁)	〃 +L	画面を消去してカーソルをホームポジションに戻す。
0D	13	CR	Carriage Return(復帰)	〃 +M	カーソルを次の行の先頭に移す。
0E	14	SO	Sift-out(シフトアウト)	〃 +N	カーソルを1項目ごと右へ移す。
0F	15	SI	Sift-in(シフトイン)	〃 +O	
10	16	DE	Data Link Escape(伝送制御拡張)	〃 +P	
11	17	D1	Device Control 1(装置制御1)	〃 +Q	
12	18	D2	Device Control 2(装置制御2)	〃 +R	[INS] カーソルの位置から右側を1文字分右へずらす。
13	19	D3	Device Control 3(装置制御3)	〃 +S	
14	20	D4	Device Control 4(装置制御4)	〃 +T	
15	21	NK	Negative Acknowledge(否定応答)	〃 +U	
16	22	SN	Synchronous idle(同期信号)	〃 +V	
17	23	EB	End of Transmission Block(伝送ブロック終了)	〃 +W	
18	24	CN	Cancel(取消し)	〃 +X	
19	25	EM	End of Medium(媒体終端)	〃 +Y	
1A	26	SB	Substitute(文字置換)	〃 +Z	
1B	27	EC	Escape(拡張)	[ESC]	実行の1時中断
1C	28	→	(FS) File Separator(ファイル分離)	[⇆]	カーソルを1つ右へ移す。
1D	29	←	(GS) Group Separator(グループ分離)	[SHIFT] +[⇆]	カーソルを1つ左へ移す。
1E	30	↑	(RS) Record Separator(レコード分離)	[⇅]	カーソルを上の行へ移す。
1F	31	↓	(US) Unit Separator(ユニット分離)	[SHIFT] +[⇅]	カーソルを下の行へ移す。

第4章　カセットファイル

第4章　カセットファイル

4-1.ファイル名に変数が使える

csave, cloadなどのファイル名として変数が使えます。ためしに、

```
a$="TEST"
csave a$
```

と行なって下さい。エラーも出ずに正常にcsaveを行ないます。正しくsaveされたか見てみましょう。

```
cload?"TEST"
```

saveミスがなければ、Found:〝TEST〟と表示されたあと、OKが表示されます。この場合もcload ? a$で構いません。

また、変数だけでなく、もっと一般的に文字式も使えます。

```
a$="TE"
cload a$+"ST"
```

どうです。ちゃんとロードできたでしょう。

4-2.データフォーマット

さて、プログラムやデータはカセットへどのような型式でsaveされるのでしょうか。

4-2-1 プログラム・ファイル

まず、csaveされたBASICのプログラム・ファイルですが、次の図を見て下さい。

〔図表4-1〕

ON	2.5秒のWAIT	「D3」×10bytes	File name(6bytes)	0.1秒のWAIT
		ヘッダー		

	データ(EB54,55)~(EFA0,A1)−1	「00」×9 bytes	1.3秒のWAIT	OFF
	プログラム本体	↑最後の3 bytesは00		

モータがONになったあと、2.5秒のWAITがあり、そのあとD3Hを10個書き込みます。ロードするときは、このD3H10個を見て、プログラムファイルの頭だと判断するわけです。

次に6バイトのファイルネームが続きます。ファイルネームが6バイトに満たないときは、後に00をつめて6バイトとします。

ここで一休み。0.1秒のWAITがあります。これはFound:…やSkip:を表示するためだと思われます。

次にやっとプログラム本体です。ここでは、EB54, 55番地に入っているプログラム開始番地から、EFA0, A1に入っている変数開始番地の1つ前までを一気に、チェックサムなどまったく付けずに書き込みます。よっぽどカセットを信頼しているのでしょうか。

プログラム本体をsaveしたあとは、これでデータが終わりだ、という印を書き込みます。00が9個です。プログラム本体の最後の3バイトは00ですから、結局00が12個続いたことになります。ロードするときは、00が10個続くと終わりだと判断します。

すべてのデータを書き込んだ後は、約1.3秒空転させたあと、モータをOFFにします。

4-2-2 データ・ファイル

次にデータ・ファイルのsaveのされ方です。データファイルはPRINT #-1を使って作成します。ファイル名はありません。

まず、プログラムファイルと同様に、WAITと、ヘッダーがありますが、ヘッダーの内容が9CHを6個となっています。

〔図表4-2〕

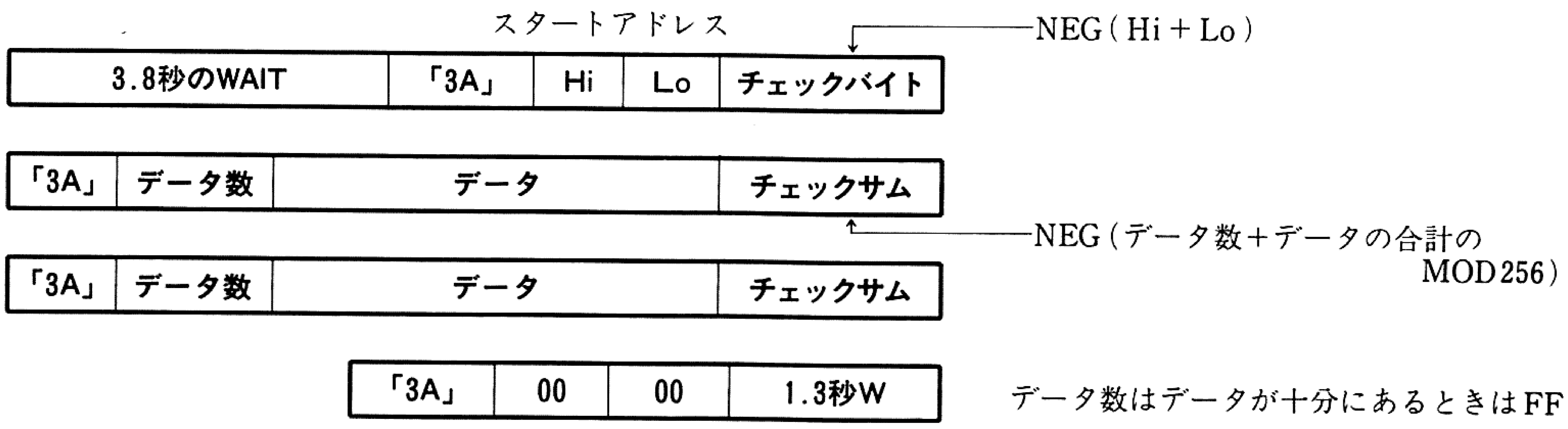
ON	WAIT	ヘッダー	データ、データ…	WAIT	OFF
		「9C」×6			

次にデータ(変数の内容)が書き込まれます。形式はCRT上に出力する時とほぼ同じですが、データの区切りのコンマは、そのまま〝,〞として書き込まれます。データを書き終わるとWAITがあって、モータがOFFになります。PRINT #-1の最後が〝;〞で終わっていても何の意味もありません。

4-2-3 モニタWコマンドのデータ形式

モニタのWコマンドはチェックサムを付けています。下図を見て下さい。

〔図表4-3〕



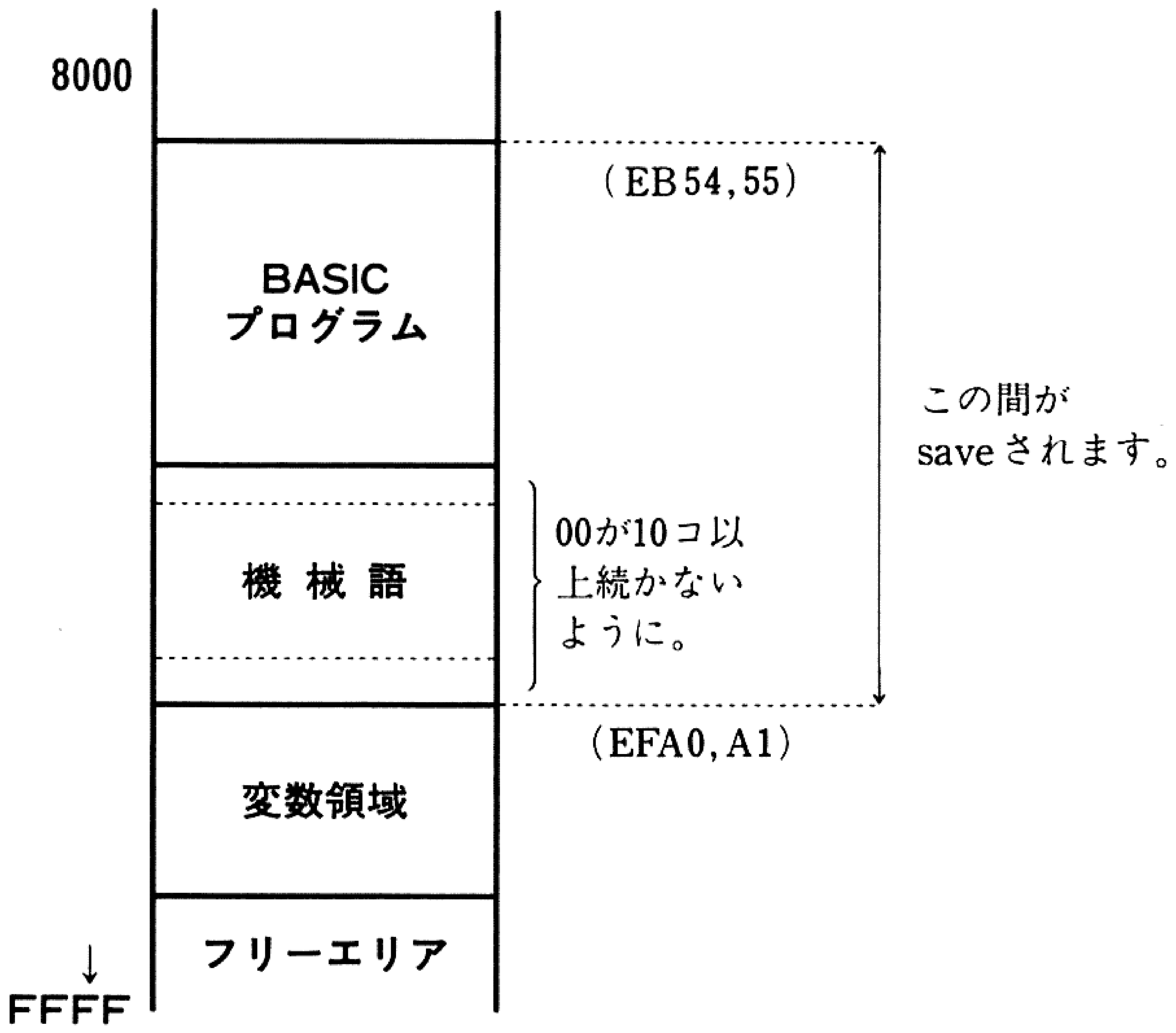
saveスタートアドレスがあって次にチェックバイトがあります。これはスタートアドレスの上位8 bitと下位8 bitを足したものを、8 bitの符号つき2進数と見なし、その符号を逆にしたもの（具体的にはNOTをとって、1足したもの)です。ここまでがヘッドブロックです。次に複数のデータブロックが続きます。(データが少ないときは1ブロックのこともあります。)各ブロックは3AH, データ数(1バイト), データ, チェックサム(1バイト)で構成されます。データ数は、普通は255個ですが、残りデータ数が255個ないときはそれ以下になります。チェックサムはデータ数と各データを足し合わせMOD256したものの符号を逆にしたものです。

データを書き終わると、データエンドの印に3A・00・00のパターンを書き込みます。つまりデータ数＝0のブロックということで、モニタのLコマンドではデータ数＝0を検出すると、終わりだと見なします。

4-3. BASICと機械語を一度にSAVE・LOAD

4-2-1でcsaveでは2つのポインタ間のデータをすべてsaveし, cloadは、00が10個現われるまでロードをすると述べました。これを利用してBASICと機械語を1度にsave・loadする方法があります。

〔図表4-4〕



まずBASICプログラムを入力します。次に(EFA0,A1)を増やして、BASICプログラムと変数領域の間に機械語を入れるのに十分な領域を作ります。そしてそこに機械語を書き込みます。ここで注意しなければならないことは、BASICのプログラムをエディットしてプログラムの長さが変わってしまった場合には、機械語の位置も変わってしまうことです。機械語自体はこわれませんが、アドレスが変わってしまうのです。十分注意しましょう。

もう一つ注意しなければならないことがあります。それは、機械語部分に続けて10個以上の00があってはならないことです。00が続けて10個以上あるとロードしたときに、そこでロードが終わってしまいます。また、最後では00が10個以上続くように、最後の3バイトぐらいは00にしておいた方がよいでしょう。

さて、機械語を書き込んだら次はsaveです。BASICだけのときと同じようにcsave"ファイル名"でsaveします。

ロードもBASICと同じようにcloadでできます。

4-4.ファイル名なしロード

cloadをするときに、ファイル名がわからなくて困ることがあります。そこでファイル名を入れなくともロードできるコマンドを作るプログラムを紹介しましょう。

```
10 '
20 '    FILENAMELESS LOAD --- POLL
30 '
40 FORI=0TO&H22:READD$:POKE&HFF50+I,VAL("&H"+D$):NEXT
50 DEFUSR=&HFF50:A=USR(0)
60 DATA 21,10,1F,11,50,F2,01,44,00,ED,B0,21,C9,00,22,5D
70 DATA F2,3E,C3,32,94,F2,21,66,1F,22,95,F2,21,50,F2,22
80 DATA 15,F1,C9
```

このプログラムは機械語を使ってあります。機械語部分のディスアセンブル結果を後に載せておきますので、機械語のわかる方は読んで見て下さい。簡単なものです。

このプログラムをrunさせると、すぐにOKが出てきます。これで、ファイル名なしのcloadコマンド「POLL」が出来ました。

カセットに何かプログラムをセットして、POLL [CR] とやって下さい。Found〝……〟と出て、プログラムをロードしてきます。

大変便利になりましたが、1つ問題があります。それはファイル名なしロードを行なうためには、このプログラムをファイル名付きでロードしてこなければならないからです。しかし、1回、runすれば、電源を切るか、リセットをかけるまでずっと使えるので少しは救われます。また、このとおり短いプログラムですから、手で打ち込んでもたいしたことはありませんし、それより機械語部分だけをモニターモードで打ち込んで、Gさせるのがよいでしょう。格納開始アドレス・実行開始アドレスともFF50番地です。

ファイル名なしロードプログラムの機械語部分は次のとおりです。

```
                ;
                ;        FILENAMELESS LOAD  M.L.
                ;
                        ORG   0FF50H
                ;
FF50 21101F             LD    HL,1F10H
FF53 1150F2             LD    DE,0F250H
FF56 014400             LD    BC,44H
FF59 EDB0               LDIR
FF5B 21C900             LD    HL,0C9H
FF5E 225DF2             LD    (0F25DH),HL
FF61 3EC3               LD    A,0C3H
FF63 3294F2             LD    (0F294H),A
FF66 21661F             LD    HL,1F66H
FF69 2295F2             LD    (0F295H),HL
FF6C 2150F2             LD    HL,0F250H
FF6F 2215F1             LD    (0F115H),HL
FF72 C9                 RET
                ;
FF73                    END
```

4-5.オートスタート

ここでは、プログラムをロードした後、自動的にrunさせる方法について述べてみましょう。

これにはファンクションキーの機能を使います。PC-8001はワークエリア内のファンクションキーフラグを立ててやると、ファンクションキーが押されたものと判断し、これまたワークエリア内のファンクションキーポインタが指すアドレスに入っている値をアスキーコードと見なしてそれを文字としてとり出し、あたかもその文字のキーが押されたかのような働きをします。

したがって、ワークエリア内のフラグ、ポインタ、そして実行すべき命令を正しくセットするようなテープを作って、それをロードすればよいのです。ではいかにしてそのテープを作るか？なぜそうなるかはさておき、その方法を下に示します。

●命令をセットする

EAC0番地から実行すべき命令を一文字ずつアスキーコードに直したものをモニタのSコマンドでセットします。命令の後には必ず00を入れます。

例：02,43,4C,4F,41,44,22,41,42,22,0D,52,55,4E,0D,00

(↑B)CLOAD"AB" CR RUN CR

●フラグをセットするために

key 1 ,"WEA68,EADF"+CHR$(13)を実行します。

●ポインタをセットするために

何もしません。

●さて、上の3つを実行したらモニターモードに入り、カセットを録音状態にして、[f･1]キーを押します。しばらくして録音が終わります。次にBASICに戻り、上の例では「AB」というファイル名で、同じテープに続けて（先ほどの録音部分の後に、という意味です。間に無録音部分があってもかまいません。）csaveします。これでできあがりです。

PCをリセットし、ファンクションキーにさわらないようにしながらモニタに入り、Lコマンドを実行します。しばらくすると、リレーがカチッとなり、CLOAD"AB"と表示されてプログラムをロードしに行きます。ロードが終わると自動的にrunがかかり、プログラムが走り始めます。

さて、どうしてこうなるのか考えてみましょう。まずPCをリセットした時には、実はファンクションキーポインタはEAC0番地を指しています。ここでテープからロードします。命令がEAC0番地からにロードされ、フラグもセットされます。したがって後はすべてうまく行く、というわけです。これでポインタをセットするために何もしなかったわけがわかったでしょう。

次になぜファンクションキーに「WEA68,…」というものをセットしたのでしょう。直接にEA68番地～EADF番地をWしてはいけないのでしょうか。……これはフラグ、つまりEA68番地を0以外の値にセットしてWするためです。フラグは、ファンクションキーにセットされた命令が実行されている間は立ったままでいます。したがって上のようにしてWするとフラグが立った状態でテープにsaveできます。

第5章　プリンタ出力

第5章　プリンタ出力

5-1.スクリーンコピーのいろいろな方法

画面のコピーをプリンタで出力したいと思うことがよくありますが、市販のスクリーンコピーROMを買おうとすると、数千円から数万円もします。

しかし、キャラクタ（グラフィックキャラクタも含む）のみの画面ならば次に示すいくつかの方法で簡単にプリンタへ出力できます。

5-1-1 ROM内のルーチンを利用

```
NEC PC-8001 BASIC Ver 1.1
Copyright 1979 (C) by Microsoft

Ok
mon
*G124A
```

124A……画面プリントルーチン

モニタモードで124A番地にジャンプするだけで、その時に画面に表示されているものがすべてプリントアウトされます。

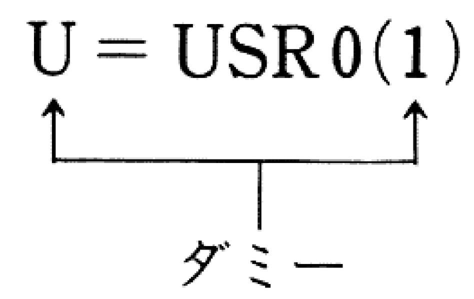
この場合、実行後、Syntax errorを表示して処理を終わります。これは、124Aから始まるルーチンがサブルーチンでありRETで戻ろうとするにもかかわらず、monのG(ジャンプ）命令で実行したため、戻り先がないためです。

機械語のサブルーチンを実行するには、USR関数があり、DEF USR文で124A番地を定義してUSR関数を実行すればエラーにならずに、画面プリントができます。

```
NEC PC-8001 BASIC Ver 1.1
Copyright 1979 (C) by Microsoft

Ok
mon
*G124A
Syntax error
Ok

defusr0=&h124a
Ok
u=usr0(1)
```



これを使ったプログラム例を次に示します。

```
100 WIDTH 40,25:DEFUSR0=&H124A:C=&H20
110 PRINT CHR$(12);TAB(4);
120 FOR I=0 TO 15:PRINT HEX$(I);" ";:NEXT:PRINT:PRINT
130 PRINT HEX$(C);"  ";
140 FOR I=0 TO 15
150 PRINT CHR$(C+I);" ";
160 NEXT:PRINT
170 C=C+16:IF C<255 THEN 130
180 PRINT:PRINT "Do you print (y/n) ?  ";:KY$=INPUT$(1)
190 IF KY$="y" THEN U=USR0(1)
```

アスキーコード(20H～FFH)を画面に表示するプログラムですが、表示後、それをプリントするかどうか聞いてきますので、"y"を入力すればUSR文を実行するので画面をプリントするというものです。

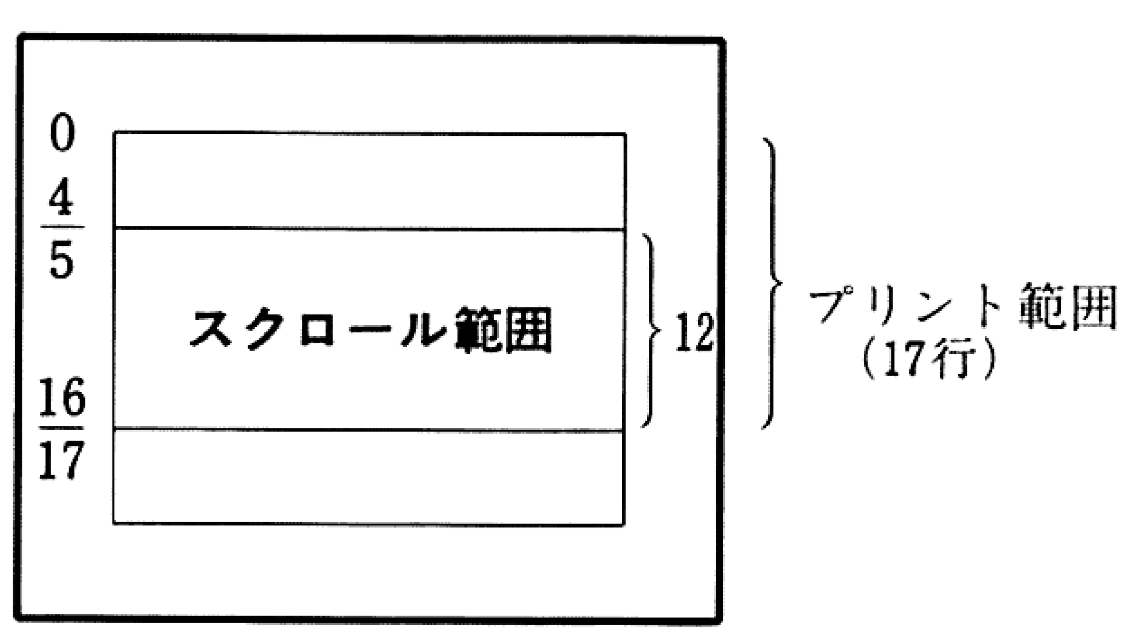
実行して、"y"を入力すると、次のようにプリントされます。

```
    0 1 2 3 4 5 6 7 8 9 A B C D E F

20    ! " # $ % & ' ( ) * + , - . /
30  0 1 2 3 4 5 6 7 8 9 : ; < = > ?
40  @ A B C D E F G H I J K L M N O
50  P Q R S T U V W X Y Z [ ¥ ] ^ _
60  ` a b c d e f g h i j k l m n o
70  p q r s t u v w x y z { ¦ } ~
80  ▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▏ ▎ ▍ ▌ ▋ ▊ ▉ ┼
90  ┴ ┬ ┤ ├ ▔ ─ │ ▕ ┌ ┐ └ ┘ ╭ ╮ ╰ ╯
A0    ｡ ｢ ｣ ､ ･ ｦ ｧ ｨ ｩ ｪ ｫ ｬ ｭ ｮ ｯ
B0  ｰ ｱ ｲ ｳ ｴ ｵ ｶ ｷ ｸ ｹ ｺ ｻ ｼ ｽ ｾ ｿ
C0  ﾀ ﾁ ﾂ ﾃ ﾄ ﾅ ﾆ ﾇ ﾈ ﾉ ﾊ ﾋ ﾌ ﾍ ﾎ ﾏ
D0  ﾐ ﾑ ﾒ ﾓ ﾔ ﾕ ﾖ ﾗ ﾘ ﾙ ﾚ ﾛ ﾜ ﾝ ﾞ ﾟ
E0  ═ ╞ ╪ ╡ ◢ ◣ ◥ ◤ ♠ ♥ ♦ ♣ ● ○ ╱ ╲
F0  ╳ 円 年 月 日 時 分 秒

Do you print (y/n) ?
```

(例) console 5, 12のとき



ところが、おわかりのようにDo you ……？のメッセージまでプリントされてしまいます。

これは、console文をプリント直前で実行することで、解決できます。

すなわち、プリント範囲は、第0行からconsole文で指定されるスクロール範囲の最下行までとなっています。

ただし、console0,25などと全部を指定しても、ファンクションキーはプリントアウトされません。

先に示したプログラムの100行、190行に下記のようにconsole文を追加すれば、0～16行がプリント範囲となり、アスキーコード表の部分のみがプリントされます。

```
100 CONSOLE 0,25:WIDTH 40,25:DEFUSR0=&H124A:C=&H20

190 IF KY$="y" THEN CONSOLE 0,16:U=USR0(1)
```

190行のconsole文の代りに、poke文により、スクロールのパラメータを直接ワークエリアのEA5DおよびEA5E番地に書込んでもかまいません。

```
190 IF KY$="y" THEN POKE &HEA5D,15:POKE &HEA5E,1:U=USR0(1)
```

EA5D………スクロールの長さ－1

EA5E………スクロールの開始行＋1

もう一つ注意しなければならないことに、スペースの処理があります。

画面をクリアキーまたは、PRINT CHR$(12)でクリアしたとき、VRAM内のデータは、すべて"00"となっています。

表示位置をTAB関数やUSING文などで指定すると、キャラクタ間は、20Hのスペースで満たされますが、locate文で指定すると、キャラクタ間は、00Hのままです。したがって、その状態で124Aからの画面プリントルーチンを実行すると、00Hをプリンタは無視するので、キャラクタ間のスペースが無視され、詰まってしまいます。

TAB指定の場合

```
100 DEFUSR0=&H124A
110 PRINT CHR$(12);"AAAAA";TAB(10);"BBBBB";TAB(20);"CCCCC"
120 CONSOLE0,1:U=US0(1):CONSOLE0,25

AAAAA         BBBBB         CCCCC
```

locate文による場合

```
100 DEFUSR0=&H124A
110 LOCATE  0,0:PRINT "AAAAA"
120 LOCATE 10,0:PRINT "BBBBB"
130 LOCATE 20,0:PRINT "CCCCC"
140 CONSOLE0,1:U=USR0(1):CONSOLE0,25
```

```
AAAAABBBBBCCCCC
```

これを防ぐためには、color文の第2パラメータを20Hにして、一旦画面クリアしておけば、V-RAMは、すべて20Hに変わるので、locate文による表示でもキャラクタ間は、20Hであり、画面プリントしてもスペースは無視されません。

```
color,&h20  または  mon               を実行後画面クリアする。
Ok                 *SEA5A
                   EA5A 00-20 00-
                   *
                   *
                   Ok
```

プログラム内に`10 COLOR ,&H20:PRINT CHR$(12)`

または、`10 POKE &HEA5A,&H20:PRINT CHR$(12)`としておいてもよい。

いずれかを実行しておけば、locate文によるプログラムを実行しても、スペースは無視されません。

5-1-2 ターミナルモードを利用

POKE文または、MONモードで、EA58番地に01Hを書込む。

```
poke&hea58,1  または  mon
Ok                   *SEA58
                     EA58 00-01 00-
                     *
                     Ok
```

上記いずれかを実行しておけば、以後、f·9キー(SHIFTキー+f·4キー)を押せば、その時、画面表示されているものがプリントアウトされます。

console文によるプリント範囲の設定、color文第2パラメータによるスペース処理については、5-1-1のROM内サブルーチンを利用する場合と全く同様です。

このモードは、f·8キー(PRINTキー)を押すことにより解除されます。したがって、このモード設定中は、f·8キーによるPRINTの画面表示はできませんのでご注意ください。

この方法による画面プリントは、例えば、プログラムlistの表示を[ESC]により中断途中であっても有効です。 (次ページ上図)

```
100 ' *** Character Print ***
110 FOR I=32 TO 255
120    PRINT I,CHR$(I);" ";
130 NEXT I
```

また、例えば、TRONモードにおいて、あるプログラムを実行中に、適当なところで、[ESC]キーにより中断し、f·9を押して、プリントアウトし、その後実行を再開することもできます。

```
tron
Ok
run
[100][110][120] 32     . [130][120] 33     ! [130][120] 34
" [130][120] 35        # [130][120] 36     $ [130][120] 37
% [130][120] 38        & [130][120] 39     ' [130][120] 40
( [130][120] 41        ) [130][120] 42     * [130][120] 43
+ [130][120] 44        , [130][120] 45     - [130][120] 46
. [130][120] 47        / [130][120] 48     0 [130][120] 49
1 [130][120] 50        2 [130][120] 51     3 [130][120] 52
4 [130][120] 53        5 [130][120] 54     6 [130][120] 55
7 [130][120] 56        8 [130][120] 57     9 [130][120] 58
: [130][120] 59        ; [130][120] 60     < [130][120] 61
= [130][120] 62        > [130][120] 63     ? [130][120] 64
@ [130][120] 65        A [130][120] 66     B [130][120] 67
C [130][120] 68        D [130][120] 69     E [130][120] 70
F [130][120] 71        G [130][120] 72     H [130][120] 73
I [130][120] 74        J [130][120] 75     K [130][120] 76
L [130][120] 77        M [130][120] 78     N
```

もちろん、通常状態でのプログラム実行途中であってもかまいません。

```
run
 32           33      !   34      "   35      #   36
$   37    %   38      &   39      '   40      (   41
)   42    *   43      +   44      ,   45      -   46
.   47    /   48      0   49      1   50      2   51
3   52    4   53      5   54      6   55      7   56
8   57    9   58      :   59      ;   60      <   61
=   62    >   63      ?   64      @   65      A   66
B   67    C   68      D   69      E   70      F   71
G   72    H   73      I   74      J   75      K   76
L   77    M   78      N   79      O   80      P   81
Q   82    R   83      S   84      T   85      U   86
V   87    W   88      X   89      Y   90      Z   91
[   92    ¥   93      ]   94      ^
```

このターミナルモードの状態において、f·10キーは、プリンタのラインフィード、そしてf·6キーは、コントロール文字の画面表示をするかしないかのスイッチの役目をもっており、当然のことながら、このモードにおいてf·6～f·10を通常のファンクションキーとして使うことはできません。

5-1-3 BASICによるサブルーチン

前述の2つの方法は、いずれも画面の第0行から任意の行までをプリントするものですが、F300番地からのVRAMの内容をPEEK関数が読み出し、プリントアウトする方法もあります。その一例を次に示します。

```
list
10000 '      ***** Screen Copy Subroutine *****
10010 '
10020 ' S: Start Line (0-24)    E: End Line (0-24)
10030 '
10040 I=&HF300+120*S
10050 FOR J=0 TO 79 STEP 80¥PEEK(&HEB4A):P=PEEK(I+J)
10060 IF P>31 AND P<248 THEN LPRINT CHR$(P);ELSE LPRINT " ";
10070 NEXT J:LPRINT:I=I+120:IF I>&HF300+120*E THEN RETURN ELSE 10050
Ok


s=0:e=11:gosub 10040
```

サブルーチン形式をとっており、Sにプリント開始行、Eに最終行を入れて、10040番地にGOSUBで行けば、SとEで指定した範囲が、スクロールなどに関係なくプリントアウトされるものです。

もし、プリンタが、グラフィックキャラクタの出ないものであれば、10060行のPの条件を変更してください。

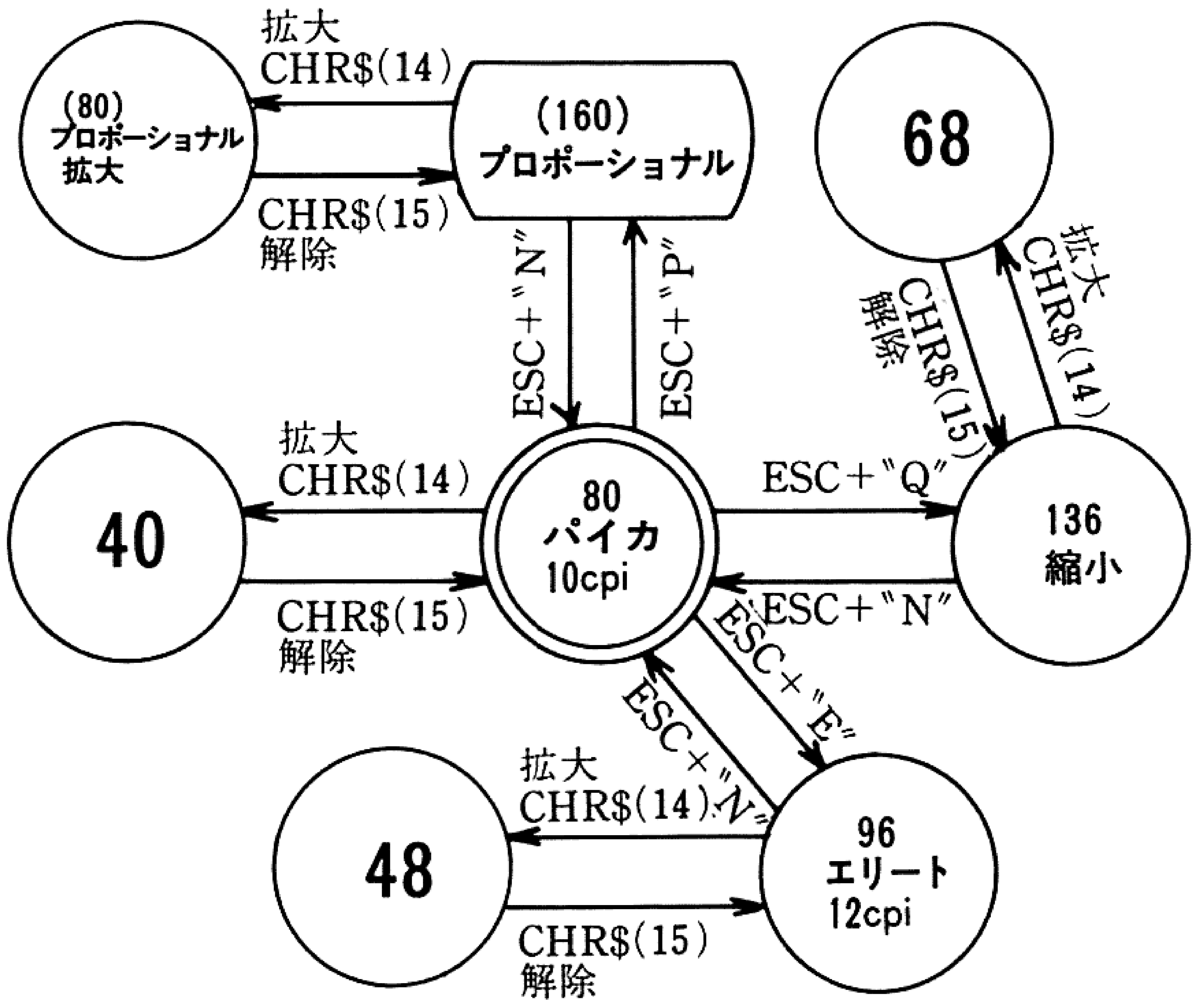
横の桁数については、EB4A番地にその内容が入っているので、80との整数除算で、40, 36桁の場合には、Step2となってVRAMを1つおきに、また80, 72桁の場合には、Step1となって連続して読み出すようになっています。

これらの画面プリント方法は、画面への表示のためのPRINT文に対応したプリンタへの出力のためのLPRINT文を各々つくる必要がなく、必要に応じ、USR文を実行したり、GOSUBによりサブルーチンをCALLするだけで良いので、プログラム作成の省力化、省メモリ化が図れます。ドットグラフィクスを含んだ画面をコピーする方法は、PC-8023の使用説明書にプログラム例がありますので、参照してください。

5-2-1 プリンタ機能一覧表 (PC-8023)

印字文字の切換え

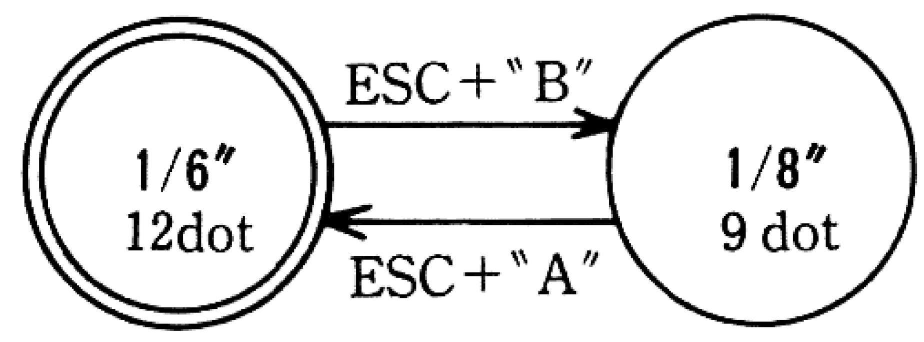
ESC：CHR$(27)
CR：CHR$(13)印字
LF：CHR$(10)印字・改行
FF：CHR$(12)ページ送り



ドットスペース

ESC+CHR$(n)　1≦n≦6

改行幅の切換え

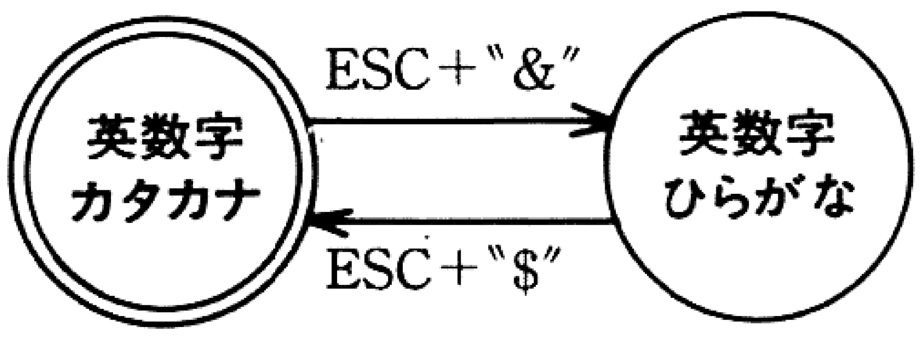


n/144″　ESC+"T″+"n_1 n_0″（2桁10進数）

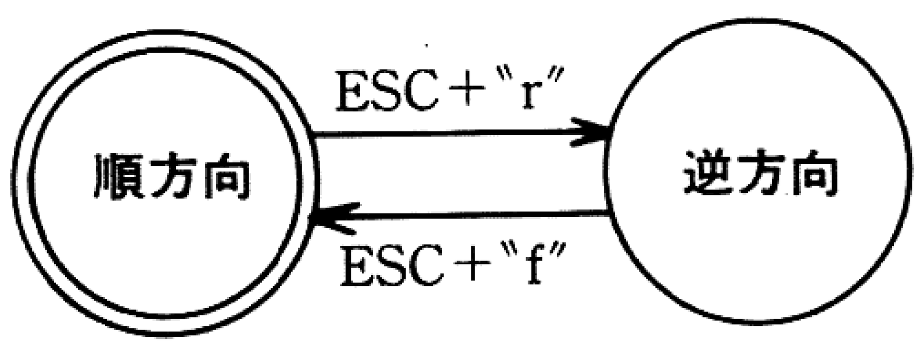
(n＝1が半ドット分に相当)

n行改行　CHR$(31)+CHR$(16+n)　1≦n≦15

キャラクタコードの切換え



改行方向切換

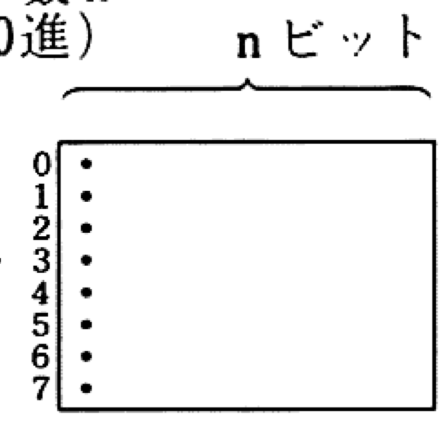


ビットイメージ

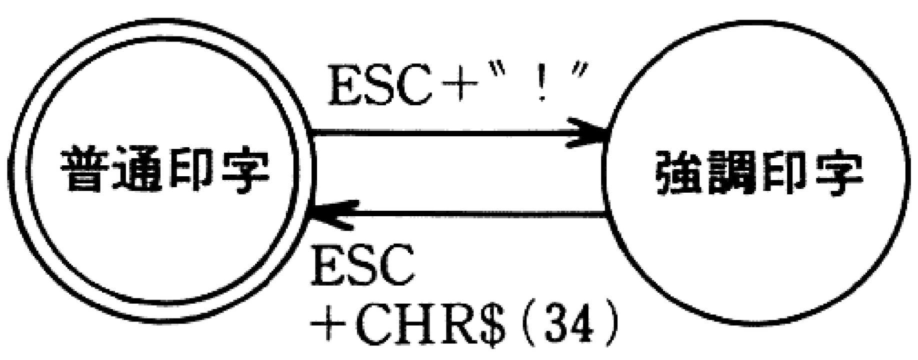
ESC+"S″+"n_3 n_2 n_1 n_0″（ビット数n（4桁10進））

〈例〉 n＝1 2 3のとき

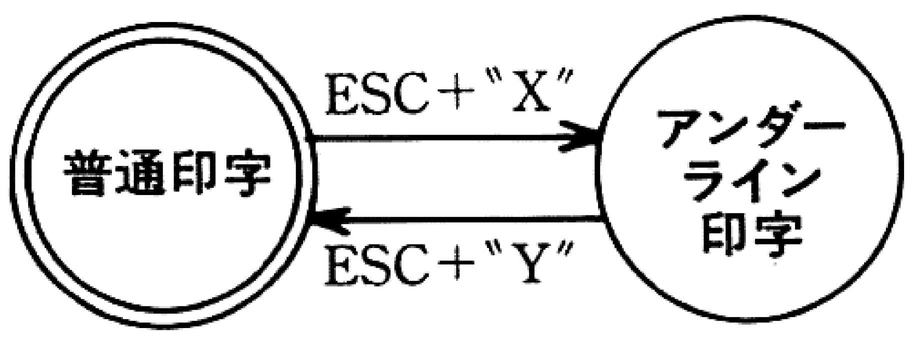
LPRINT CHR$(27)"S0123″



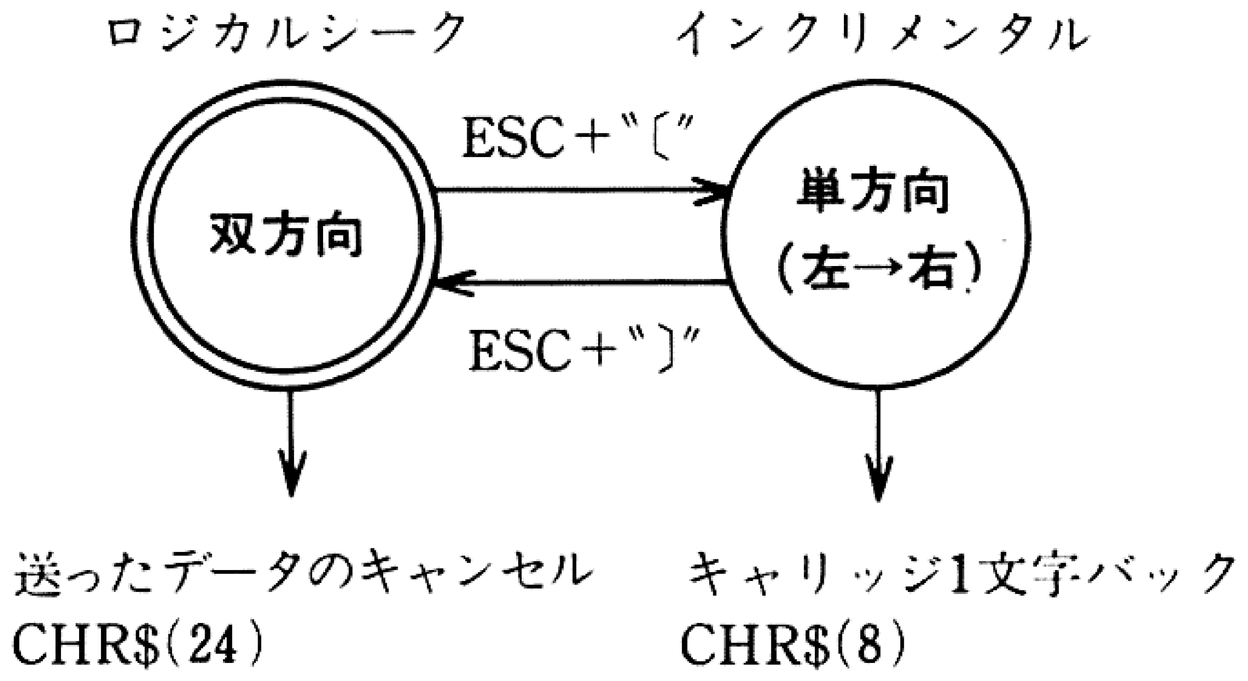
強調印字



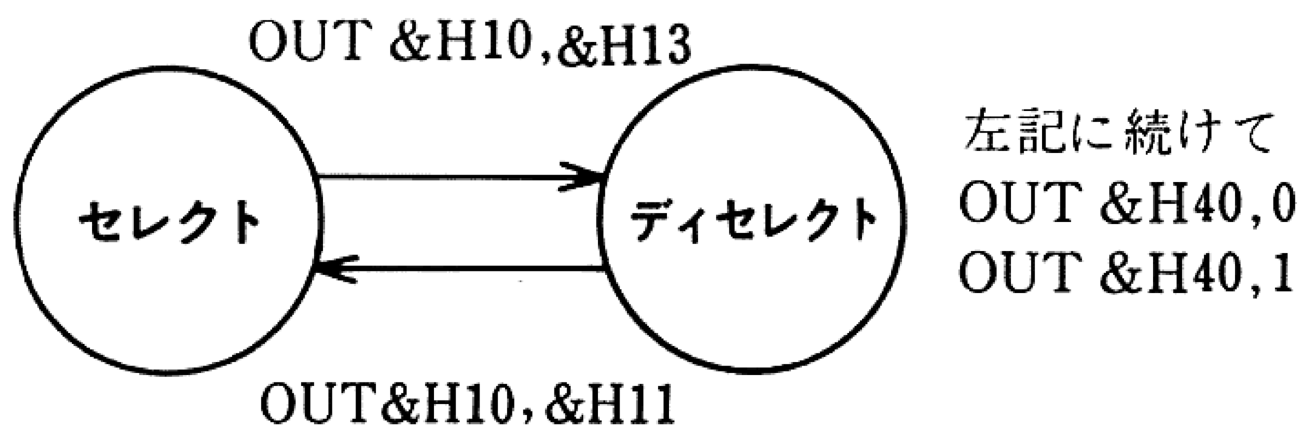
アンダーライン印字



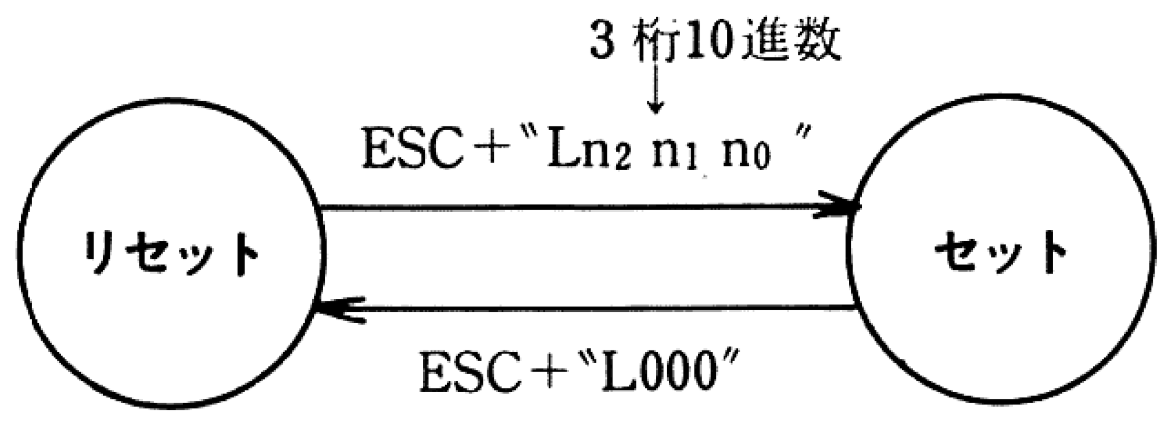
印字方向



セレクト／デイセレクト

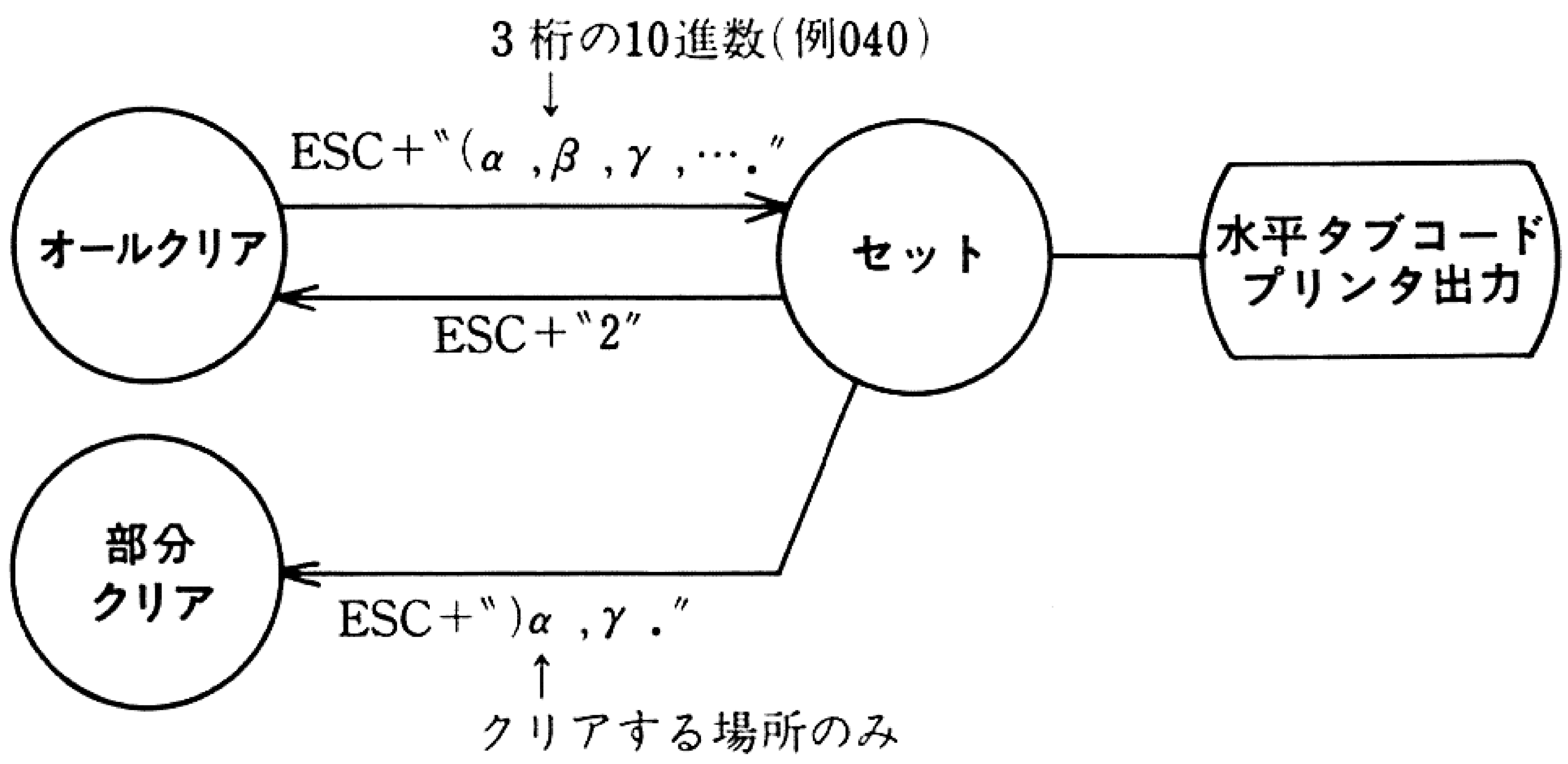


レフトマージン

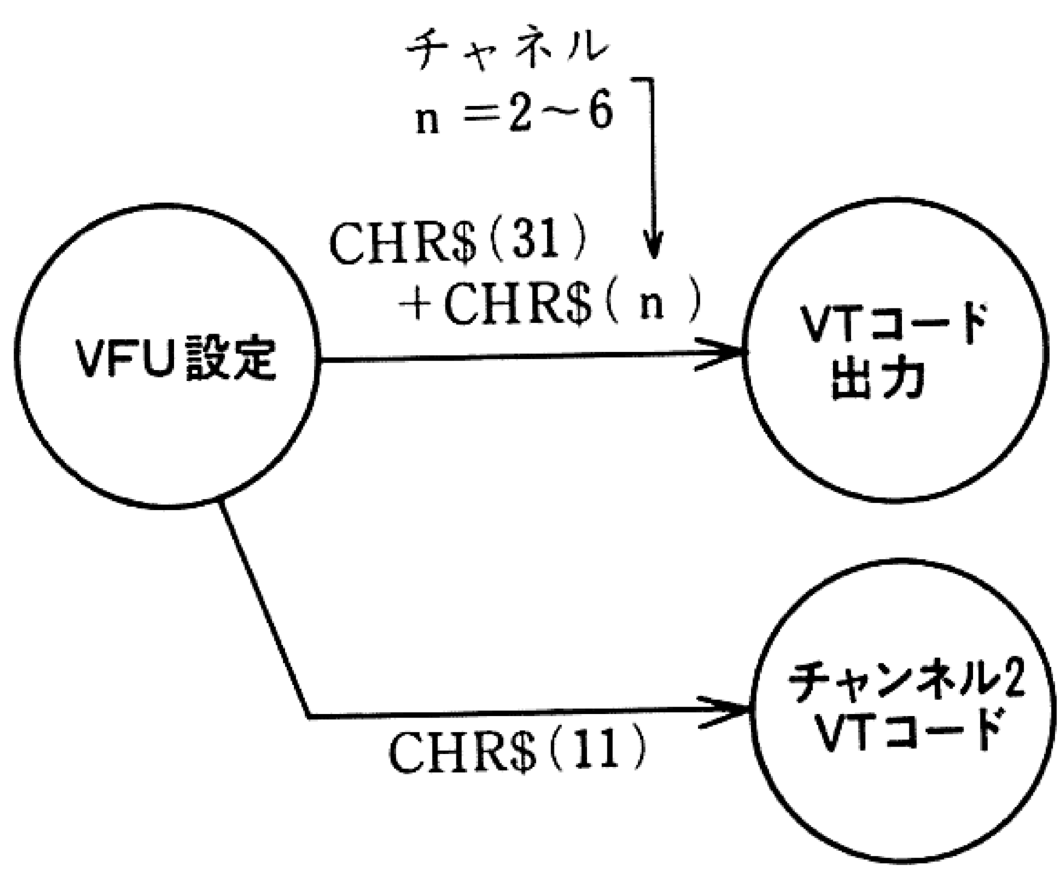



```
100 OUT &H10,9
110 OUT &H40,0
120 OUT &H40,1
130 A=INP(&H40)AND1
140 1F A=0 THEN RETURN ELSE130
```

水平タブ



垂直タブ



					CH6	CH5	CH4	CH3	CH2	CH1	
			7	6	5	4	3	2	1	0	
VFU設定	CHR$(29)		0	0	0	1	1	1	0	1	GSコード
	"A"	1	×	1	0	0	0	0	0	1	TOF
	"@"		×	1	×	×	×	×	×	×	
	"@"	2	×	1	0	0	0	0	0	0	
	"@"		×	1	×	×	×	×	×	×	
	⋮	⋮									
	"C"	60	×	1	0	0	0	0	1	1	BOTTOM
	"@"		×	1	×	×	×	×	×	×	
	"A"	67	×	1	0	0	0	0	0	1	TOF
	"@"		×	1	×	×	×	×	×	×	
	CHR$(28)		0	0	0	1	1	1	1	0	RSコード

DIP スイッチ機能

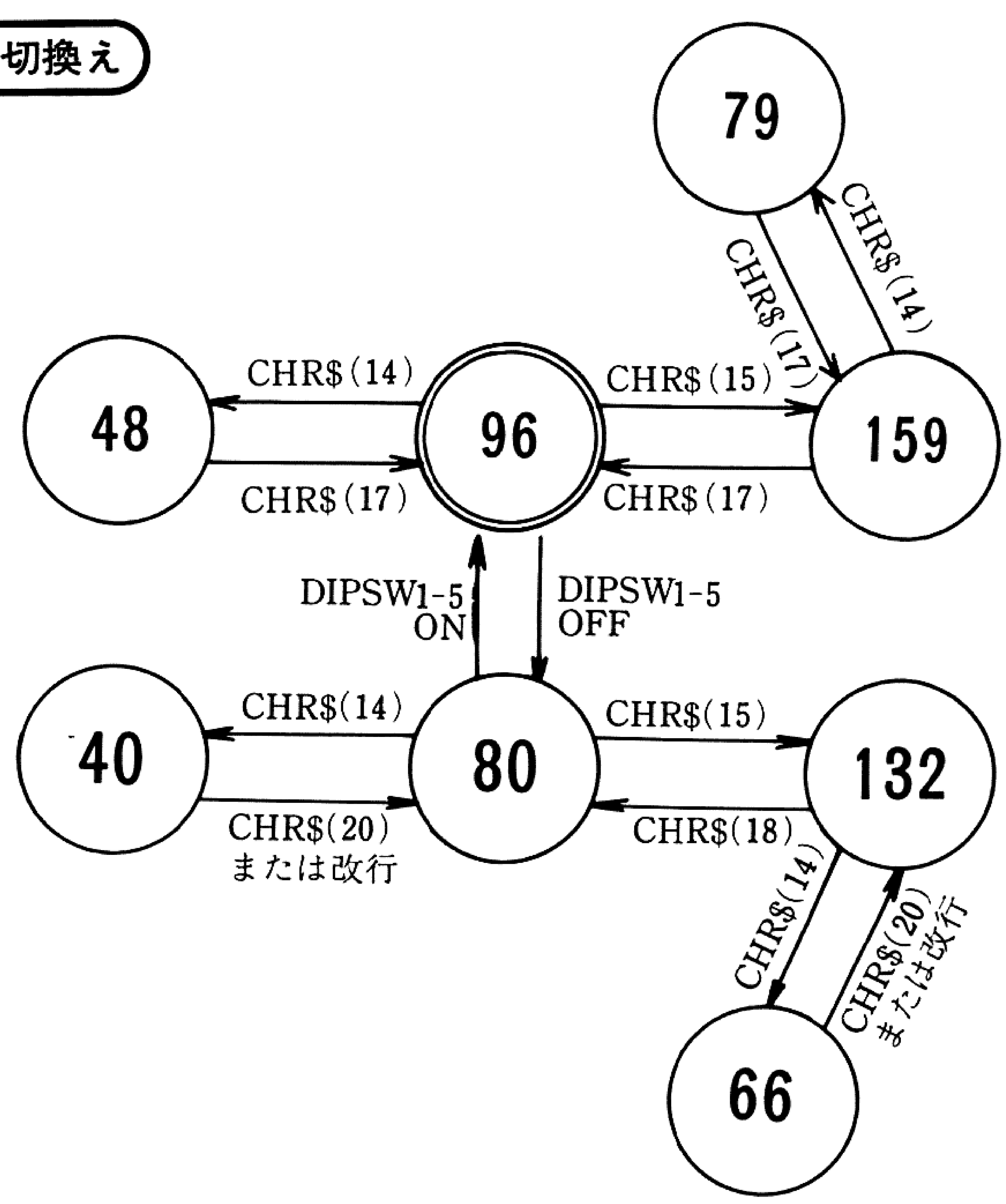
		SW1-1	SW1-2	SW1-3	SW1-4	SW1-5	SW1-6	SW1-7	SW1-8
	国別仕様切換				TOF間	DC1,3処理	1ラインフル 印字でLF有無	印字指令 コード	CRの機能
ON	JA	×	×	×	72行	無効	LF有	CR LF VF FF	CR + LF
	US	×	○	×					
OFF	UK	○	○	×	66行	有効	LF無	CRのみ	CR
	GE	×	×	○					
	SW	○	×	○					

	SW2-1	SW2-2	機器No.	SW2-3	SW2-4	SW2-5	SW2-6	SW2-7
	数字0表示	機器アドレス	機器アドレス設定			電源ON時 印字モード	7/8ビット切換	電源ON時 セレクト/ディセレクト
ON	Ø	有効	0	×	×	プロポーショナル	7 bit	セレクト
			1	○	×			
OFF	0	無効	2	×	○	パイカ	8 bit	ディセレクト
			3	○	○			

プリンタの選択
ESC＋"a"～"d"
0 ～ 3
セレクト解除
ESC＋CHR$(96)

5-2-2 プリンタ機能一覧表 (MP-82 TYPE II)

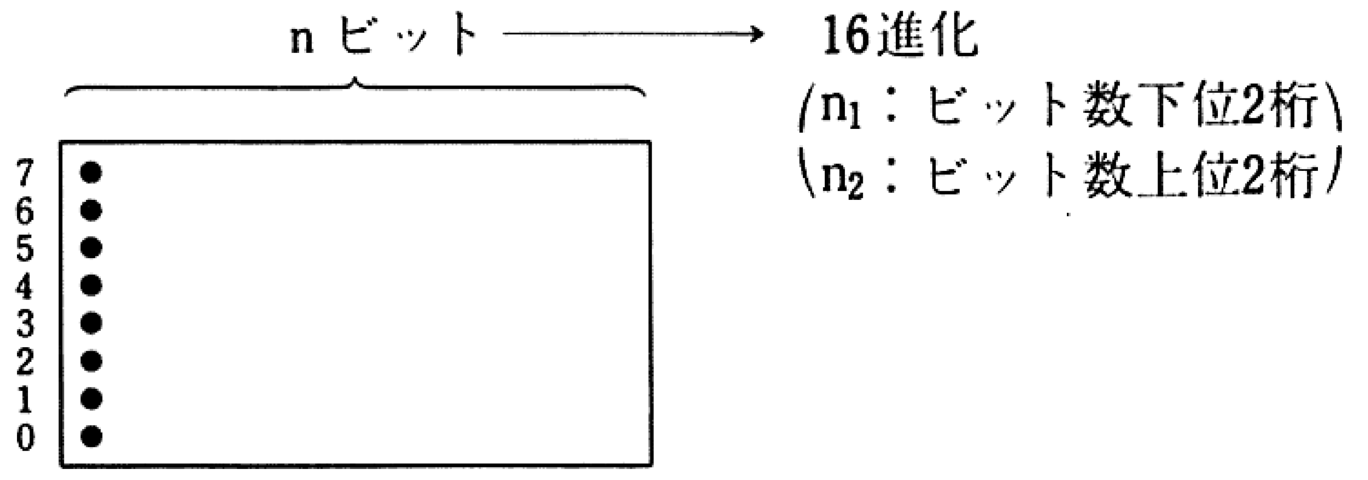
印字文字の切換え



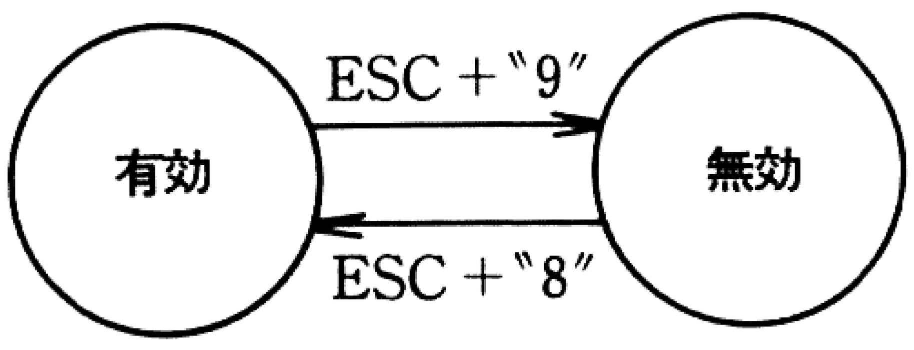
ビットイメージ

ビットイメージは単方向印字

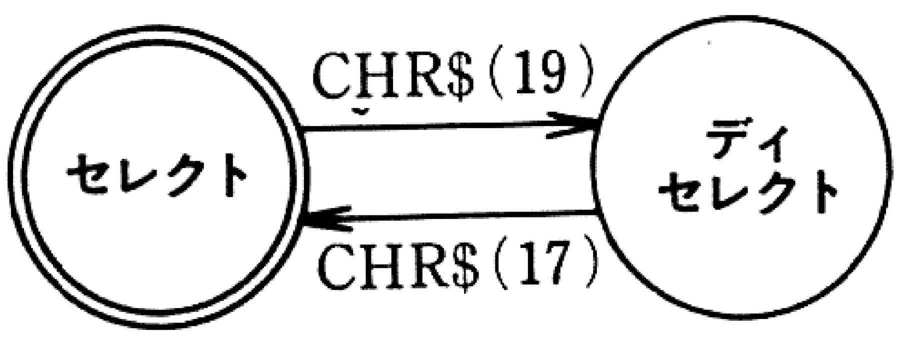
標準密度 576ビット/行	ESC＋"K"＋CHR$($n_1$)＋CHR$(n_2)
倍密度 1152ビット/行	ESC＋"L"＋CHR$($n_1$)＋CHR$(n_2)



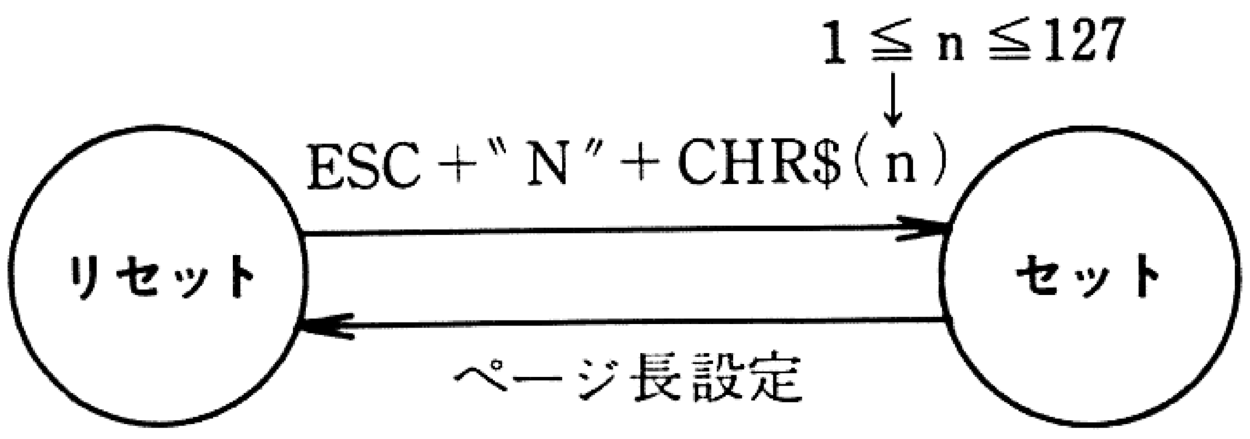
紙検出コントロール



セレクト／ディセレクト

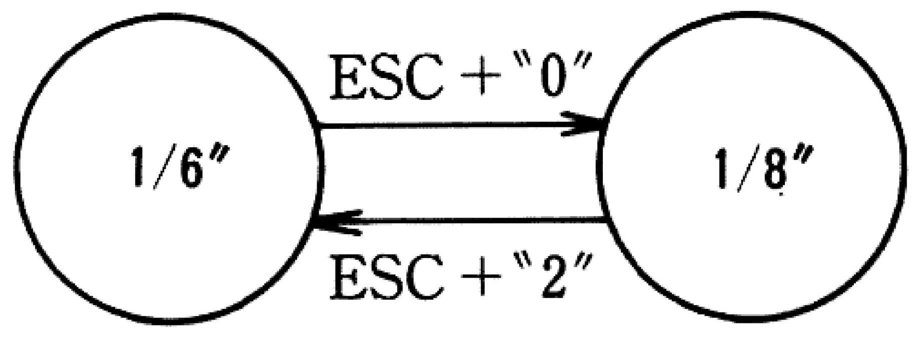


スキップパーフォレーション



ESC：CHR$(27)
CR：CHR$(13) 印字
LF：CHR$(10) 印字・改行
FF：CHR$(12) ページ送り
VT：CHR$(11) LFに同じ

改行幅の切換え



n/72"	ESC＋"A"＋CHR$(n)　(1≦n≦85)

ページ長設定

行数で設定	ESC＋"C"＋CHR$(n) (n行)　(1≦n≦127)
長さで設定	ESC＋"C"＋CHR$(0)＋CHR$(m) (m：inch)　(1≦m≦22)

ベル

CHR$(7)

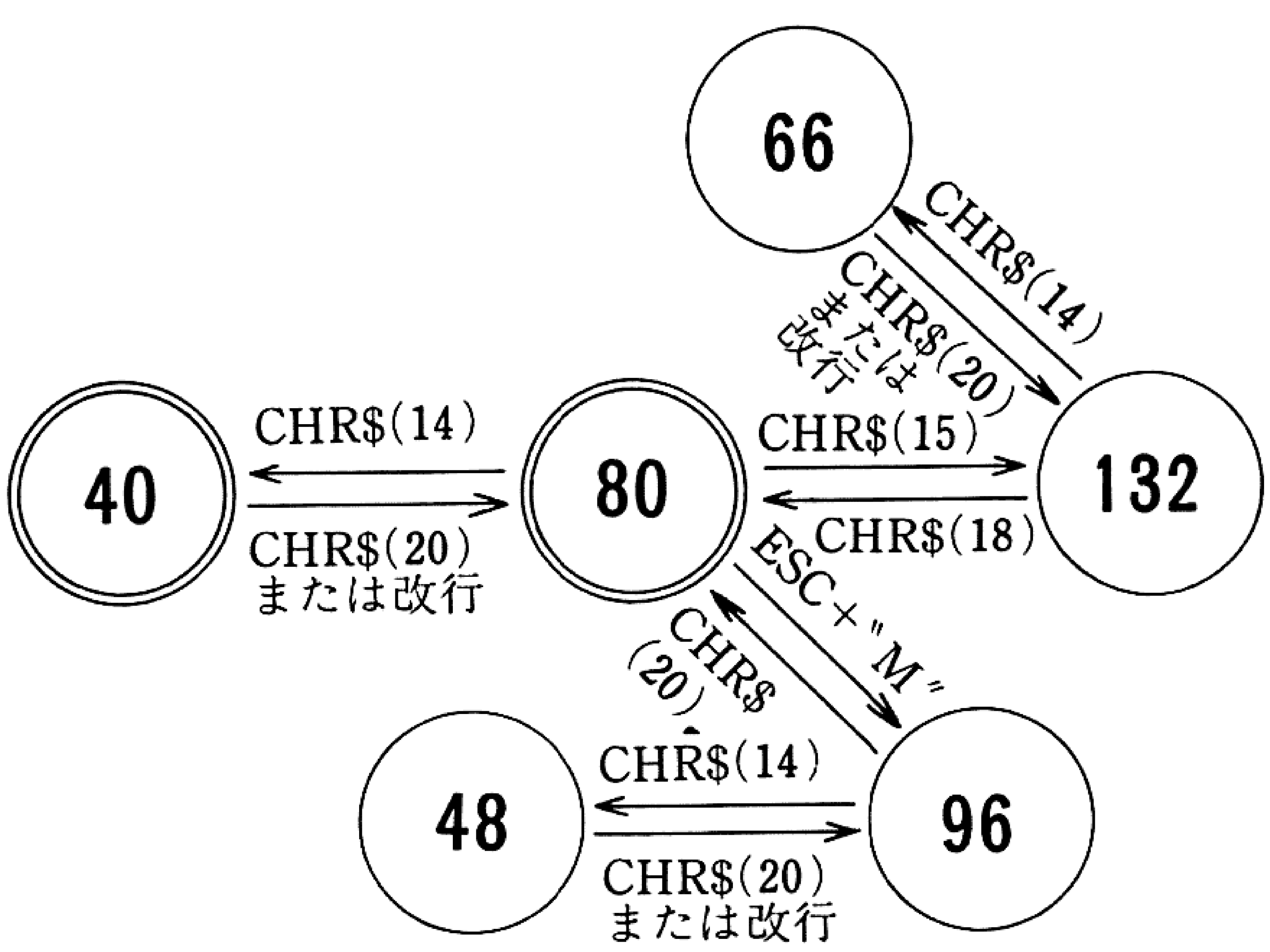
DIPスイッチ機能

	SW1-1	SW1-2	SW1-3	SW1-4		SW1-5	SW1-6	SW1-7	SW1-8
	改行幅	ページ長	1行の桁数			1行印字幅	紙検出	国別仕様切換	リモートセレクト
			SW1-3	SW1-4	桁数				
ON	1/8″	12″	×	×	80/96	8″	無効	輸出向	内部固定する
			×	○	132/159				
OFF	1/6″	11″	○	×	—	6.7″	有効	日本向	内部固定しない
			○	○	132/159				

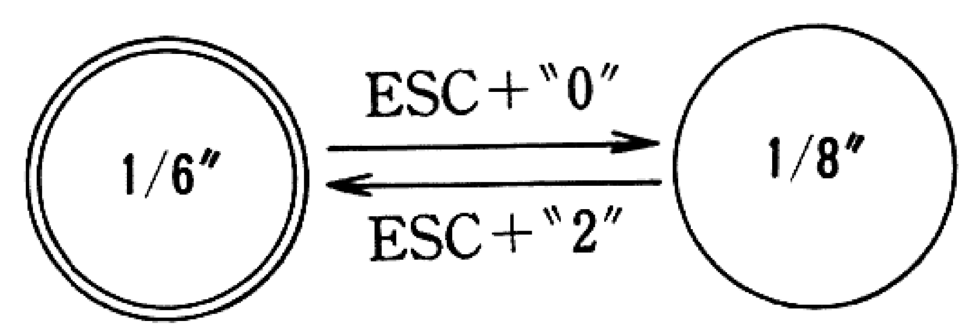
	SW2-1	SW2-2		SW2-3	SW2-4
	国別キャラクタゼネレータ切換			自動改行	1インチ自動スキップハーフオレーション
ON	1	2	仕向地	内部固定する	無効
	OFF	OFF	イギリス		
	OFF	ON	ドイツ		
OFF	ON	OFF	フランス	内部固定しない	有効
	ON	ON	アメリカ		

5-2-3. プリンタ機能一覧表 (MP-80 TYPE II)

印字文字の切換え



改行幅の切換え



n/72″　ESC+"A″+CHR$(n)
(1≦n≦85)

ページ長設定

行数で設定　ESC+"C″+CHR$(n)
(1≦n≦127)

長さで設定　ESC+"C″+CHR$(0)
+CHR$(m)
(1≦m≦22 m:インチ)

ビットイメージ

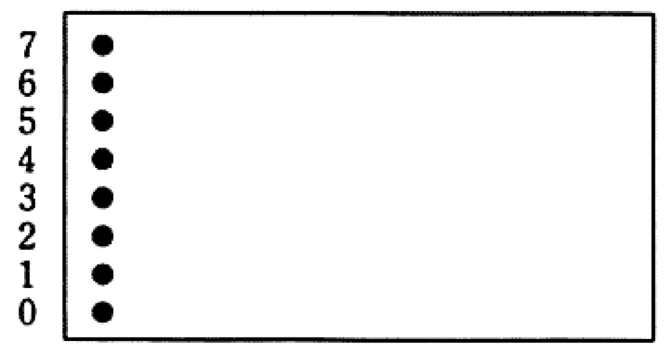
標準密度 480ビット/行　ESC＋"K"＋CHR$(n1)＋CHR$(n2)

倍密度 960ビット/行　ESC＋"C"＋CHR$(n1)＋CHR$(n2)

n1: 16進下位2桁 n mod256

n2: 16進上位2桁 n ￥256

nビット



スキップパーフォレーション

リセット →（ESC＋"N"＋CHR$(n)）→ セット　n行スキップ（1≦n≦127

セット →（ページ長設定）→ リセット

セレクト／ディセレクト

セレクト →（CHR$(19)）→ ディセレクト

ディセレクト →（CHR$(17)）→ セレクト

紙検出コントロール

有効 →（ESC＋"8"）→ 無効

無効 →（ESC＋"9"）→ 有効

その他●BL(ベル) ……………………CHR$(7):ブザーを1秒間鳴らす。
●BS(バック・スペース)………CHR$(8):バッファに入力された直前のデータキャンセ
●LF(ラインフィード)……… CHR$(10):印字、改行
●VT(ヴァーチカルタブ)…… CHR$(11):LFに同じ
●FF(フォーム・フィード)……CHR$(12):ページ送り(電源ON時の位置まで紙送り)
●CR(キャリッジ・リターン)…CHR$(13):印字

DISスイッチ機能

SW 1	1	2	3	4	桁数	5	6	7	8
	改行幅	ページ長	1行の桁数				紙検出		リモートセレクト
ON	1/8"	12"	OFF	OFF	80		無効		内部固定する
			OFF	ON	132				
OFF	1/6"	11"	ON	OFF	96	常時OFF	有効	常時OFF	内部固定しない
			ON	ON	132				

SW 2	1	2	3	4
			自動改行	1インチ自動スキップパーフォレーション
ON			内部固定する	無効
OFF	常時OFF	常時OFF	内部固定しない	有効

注1) ◎、□ は、標準状態を示します。
注2) ESCとは、CHR$(27)を意味します。

5-3.LPRINTサブルーチン

PC-8023をはじめ、最近のプリンタは、前述の機能一覧表を見てもわかるように、インテリジェントプリンタとして数多くの機能を持っていて、それを覚えて使いこなすことは並たいていのことではありません。

そこで、その機能をサブルーチンとしておき、gosub文だけでそのほとんどの機能を活用するという方法もあります。

機能一覧表代りにして、機能を覚えるのにも役立つと思います。

PC-8023を例にしたプログラム例を次に示します。

```
10000 '      *****  Lprint Subroutine Set  *****
10010 LPRINTCHR$(14);:RETURN                    'Enlarge
10020 LPRINTCHR$(15);:RETURN                    'Normal
10030 LPRINTCHR$( 8);:RETURN                    'Back space (Incremental  mode)
10040 LPRINTCHR$(18);:RETURN                    'Data cancel(Logical seek mode)
10050 LPRINTCHR$(27)"N";:RETURN                 'Pica            80/line
10060 LPRINTCHR$(27)"E";:RETURN                 'Elite           96/line
10070 LPRINTCHR$(27)"Q";:RETURN                 'Condensed      136/line
10080 LPRINTCHR$(27)"P";:RETURN                 'Proportional   160/line
10090 LPRINTCHR$(27)"!";      :RETURN           'Enhanced
10100 LPRINTCHR$(27)CHR$(34);:RETURN            'Enhanced Cancel
10110 LPRINTCHR$(27)"X";:RETURN                 'Underline
10120 LPRINTCHR$(27)"Y";:RETURN                 'Underline Cancel
10130 LPRINTCHR$(27)"A";:RETURN                 '1/6"  Line feed
10140 LPRINTCHR$(27)"B";:RETURN                 '1/8"  Line feed
10150 LPRINTCHR$(27)"T"N$;:RETURN               'n/144"Line feed  n$..2 digits
10160 LPRINTCHR$(31)CHR$(16+N);:RETURN          'N lines    feed (1=<N<=15)
10170 LPRINTCHR$(27)"$";:RETURN                 'ANK
10180 LPRINTCHR$(27)"&";:RETURN                 'ANH
10190 LPRINTCHR$(27)"f";:RETURN                 'Forward Line feed
10200 LPRINTCHR$(27)"r";:RETURN                 'Reverse Line feed
10210 LPRINTCHR$(27)"[";:RETURN                 'Incremental
10220 LPRINTCHR$(27)"]";:RETURN                 'Logical seek
10230 LPRINTCHR$(27)CHR$(N);:RETURN             'Dot space (1=<N<=6)
10240 LPRINTCHR$(27)"L"N$;:RETURN               'Left margin n$..3 digit
10250 LPRINTCHR$(27)"S"N$;:RETURN               'Bit image   n$..4 digit
10260 '      Horizontal tab
10270 LPRINTCHR$(27)"("N$".";:RETURN            'Tab set    n$..a,b,c,...,f
10280 LPRINTCHR$(27)")"N$".";:RETURN            'Tab reset  n$..b,...,e
10290 LPRINTCHR$(27)"2";      :RETURN           'Tab clear
10300 '      Vertical tab
10310 N$=STRING$(116,"@")+"C"+STRING$(13,"@")
10320 LPRINTCHR$(29)"A@"N$"A@"CHR$(30);:RETURN'Tab set  n$=string$(130,"@")
10330 LPRINTCHR$(31)CHR$(N);            :RETURN'Select   (1=<N<=6)
10340 LPRINTCHR$(11);                   :RETURN'VT code
10350 '      Lptchr                                u=usr(c%)
10360 RESTORE10390:FORI=0TO6:READA$
10370 POKE&HFF60+I,VAL("&h"+A$):NEXT
10380 DEFUSR=&HFF60:RETURN
10390 DATA f5,7e,cd,2b,00,f1,c9
10400 '      Lptchr
10410 OUT&H10,C%:OUT&H40,0:OUT&H40,1
10420 IF INP(&H40)AND1=0 THEN RETURN ELSE 10420
```

次に説明するもの以外は、すべてGOSUB文のみで使用できます。

10150行…n/144インチ改行

n$に2桁の10進数文字列を入れてGOSUB

例　n$="08"：GOSUB 10150

10160行…N行改行（1≦N≦15）

Nに数値を入れてGOSUB

10230行…Nドットスペース（1≦N≦6）

Nに数値を入れてGOSUB

10240行…レフトマージン

n$に3桁の10進数文字列を入れてGOSUB

例　n$="005"：GOSUB 10240

10250行…ビットイメージ印字

n$に4桁の10進数文字列を入れてGOSUB

例　n$="0200"：GOSUB 10250

10270行…水平タブセット

n$にタブ位置桁数並びを入れてGOSUB

例　n$="10,20,30,40,60"：GOSUB 10270

10280行…水平タブリセット

n$にキャンセルしたいタブ位置桁数並びを入れてGOSUB

例　n$="20,40"：GOSUB 10280

10320行…垂直タブセット

n$に2～66行を設定する文字列を入れてGOSUBしますが、10310行にGOSUBすれば、n$に60行をBOTTOM行とする設定データを入れて実行しますので、GOSUBの前にn$に文字列を入れる必要はありません。

10330行…垂直タブチャンネルセレクト

Nに使用するチャンネルを入れてGOSUB

10360行…1文字プリンタ出力用の機械語サブルーチンを書込むサブルーチンで、プログラムの最初で実行し、使用法は、USR文の引数に出力するコードを整数で入れて実行します。

サブルーチン書込み…GOSUB 10360

プリンタ出力…………U=USR(C%)　　C%：単精度変数

10410行…1文字プリンタ出力

C%に出力データを入れてGOSUB

例　C%=9：GOSUB 10410

参考までに、MP-80のサブルーチン例も紹介いたします。

```
10000 '     ***** Lprint Subroutine Set for MP-80 *****
10010 LPRINT CHR$(14);:RETURN                       ' Enlarge
10020 LPRINT CHR$(20);:RETURN                       ' Enlarge cancel
10030 LPRINT CHR$(15);:RETURN                       ' Condensed   132/line
10040 LPRINT CHR$(18);:RETURN                       ' Condensed cancel
10060 LPRINT CHR$(27);"M";:RETURN                   ' Elite ( 96/line )
10070 LPRINT CHR$(20);:RETURN                       ' Elite cancel
10080 LPRINT CHR$(27);"A";CHR$(7);:RETURN           ' 1/6 Line feed
10090 LPRINT CHR$(27);"A";CHR$(8);:RETURN           ' 1/8 Line feed
10100 LPRINT CHR$(27);"A";CHR$(N);:RETURN           ' N/72 ( 1<=N<=85 )
10110 LPRINT CHR$(27);"C";CHR$(N);:RETURN           ' Page length  ( 1<=N<=127 )
10120 LPRINT CHR$(27);"C";CHR$(0);CHR$(N);:RETURN     ' Page length
                                                        ( 1<=N<=22  N:Inch )
10140 LPRINT CHR$(19);:RETURN                       ' Diselect
10150 LPRINT CHR$(17);:RETURN                       ' Select
10160 LPRINT CHR$(7);:RETURN                        ' BL( Bell )
10170 LPRINT CHR$(8);:RETURN                        ' BS( Back Space )
10180 LPRINT CHR$(10);:RETURN                       ' LF( Line Feed )
10190 LPRINT CHR$(11);:RETURN                       ' VT( Vertical Tab : = LF )
10200 LPRINT CHR$(12);:RETURN                       ' FF( Form Feed  : Page End )
10210 LPRINT CHR$(13);:RETURN                       ' CR( Carriage Return : Print Out )
10220 LPRINT CHR$(27);"8";:RETURN                   ' Paper Detect OFF
10230 LPRINT CHR$(27);"9";:RETURN                   ' Paper Detect ON
10240 LPRINT CHR$(27);"K";CHR$(NN);CHR$(MM);  ' Bit Image Print Normal (480/line)
10250 FOR I=1 TO NN:LPRINT CHR$(LL);:NEXT:RETURN
10260 LPRINT CHR$(27);"L";CHR$(NN);CHR$(MM);  ' Bit Image Print Double (960/line)
10270 FOR I=1 TO NN:LPRINT CHR$(LL);:NEXT:RETURN
```

第6章　ディスクファイル

6−1　ディスクファイルの構造
6−2　ディスク使用時のユーザーズエリア
6−3　ディスク用ダンプリスト
6−4　ディレクトリ
6−5　ID
6−6　FAT(File Allocation Table)
6−7　ファイルのLocation
6−8　ファイルの転送
6−9　データディスクをシステムディスクに
6−10　Disk BASICの回復法・切離し法
6−11　ファイルのLOAD法
6−12　ディスク内データの直接修正
6−13　シークレットファイル
6−14　FIELD文とファイルデザインシート
6−15　ディスクの両面使用法
6−16　ファイルネームの並べ替え

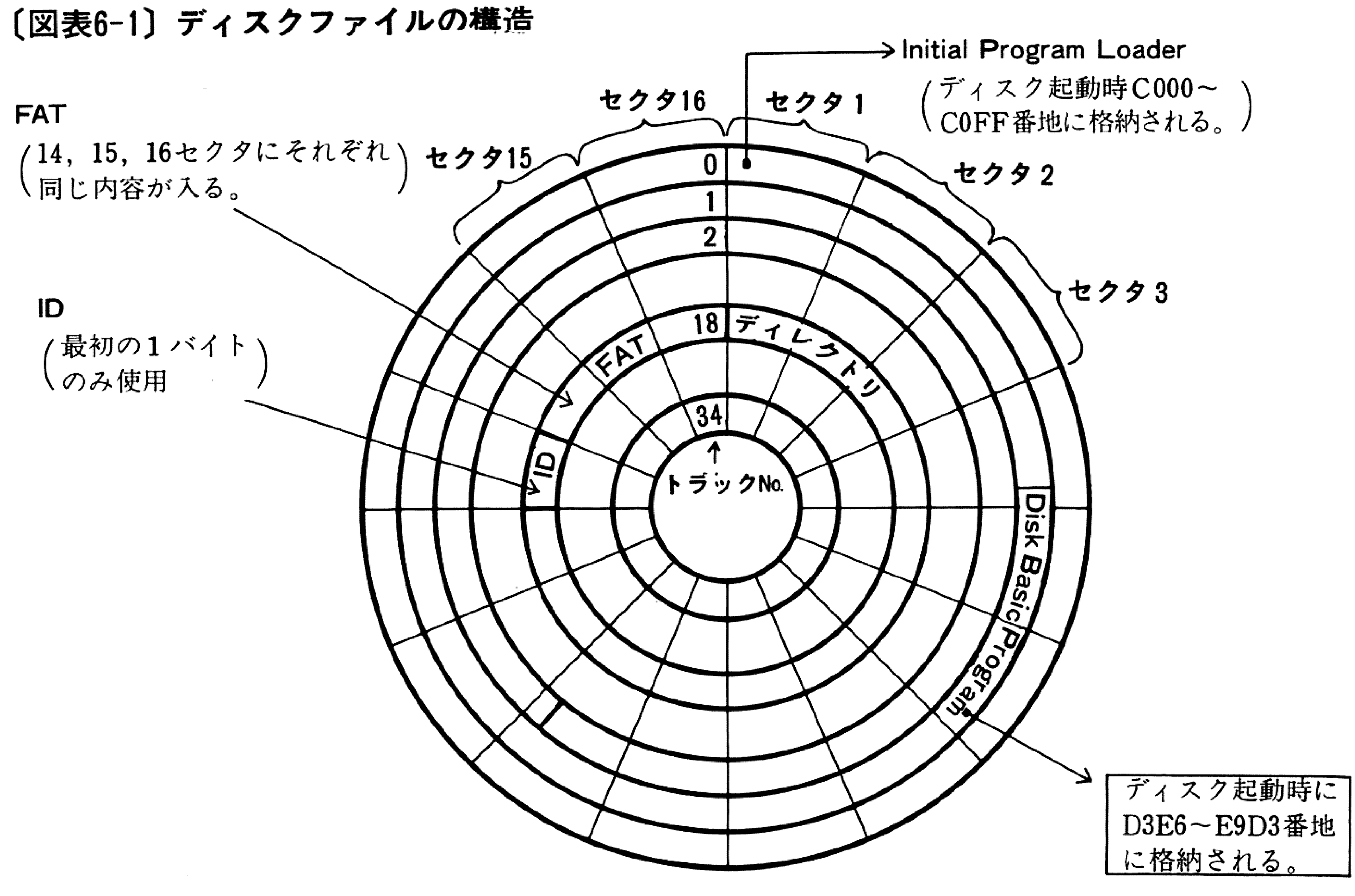
第6章　ディスクファイル

6-1.ディスクファイルの構造

ディスクファイルの構造は、図のようになっています。システムディスクの場合、第0トラックの第1セクタに、IPL(Initial Program Loader)が入っておりディスクシステム起動時に第0～2トラックの内容(12Kバイト)が、そっくりC000～EFFF 番地に転送されます。したがって、第1トラックの第4セクタの終わりの部分から、第2トラックの第10セクタまでの、Disk BASIC Programは、D3E6～E9D3番地に転送されます。

また、第18トラックには、第1セクタから第12セクタにディレクトリ（ファイル名や格納場所および属性を記録する部分)、第13セクタの最初の1バイトにID(ディスケットの属性)、第14～16セクタにFAT(File Allocation Table……ファイルの格納場所と順序を記録する部分）が各70バイトづつ同じものが入っています。

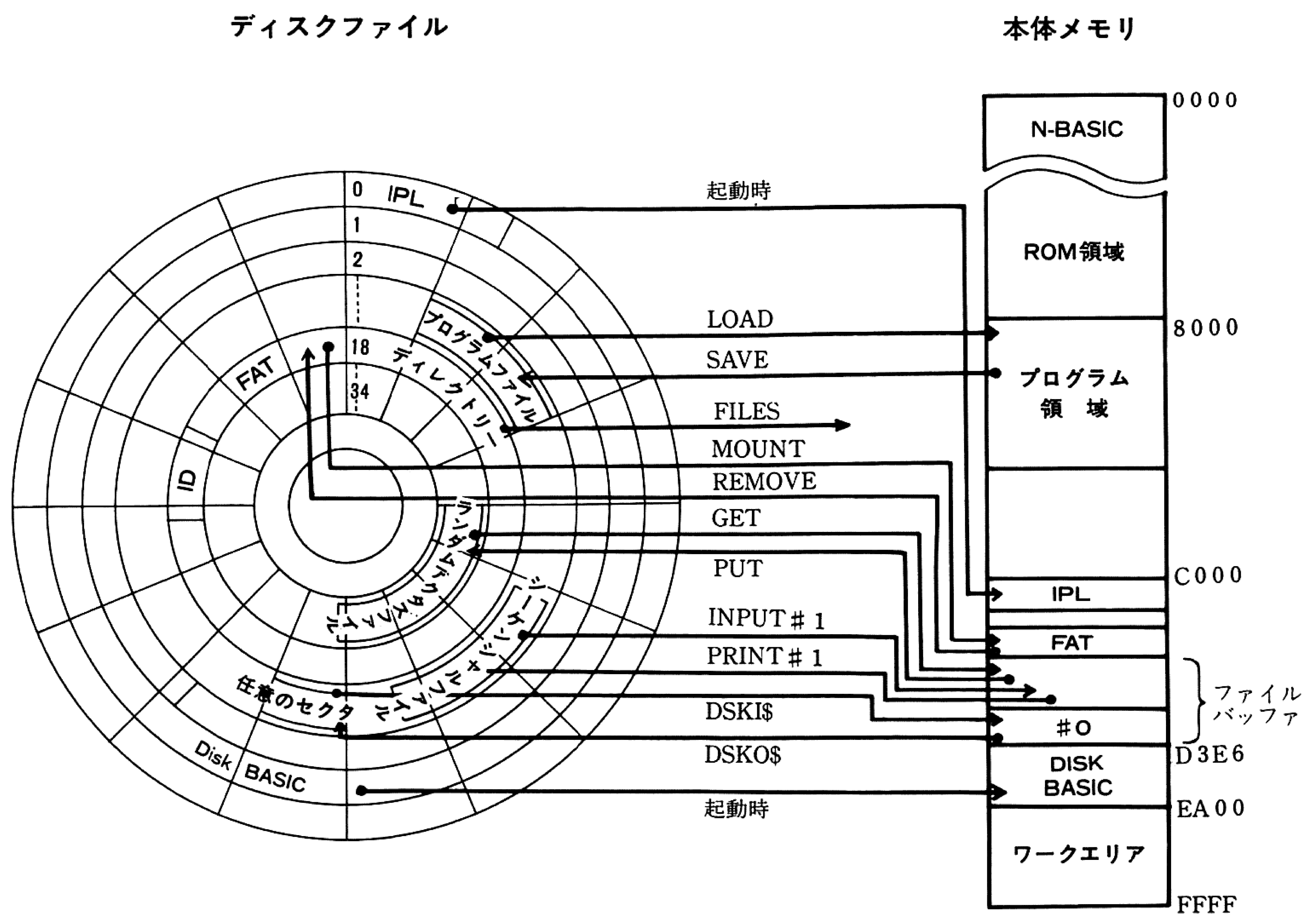
〔図表6-1〕ディスクファイルの構造



〔図表6-2〕ディスクファイルの容量

ドライブ	トラック	クラスタ	セ ク タ	バ イ ト
			1	256
		1	8	2,048
	1	2	16	4,096
1	35	70	560	143,360
		ファイル管理の単位	入出力の単位	データの単位

〔図表6-3〕Disk BASIC 入出力関係図



注1） この章ではPC-8031についてのみ記述されています。

注2） FILES, LFILESの実行により、ディレクトリの内容がバッファ＃0を介して、画面やプリンタに出力される。

注3） SAVE, KILL, PRINT＃1, PUTなどのコマンドでDiskファイルの内容を書き変えた時、本体メモリ内のFATのみが書き変えられ、ディスクファイル内のFATは、元のままですので、ディスクファイルを取り出すときは、必らずREMOVEをしてディスクファイル内のFATを更新することを忘れずに！

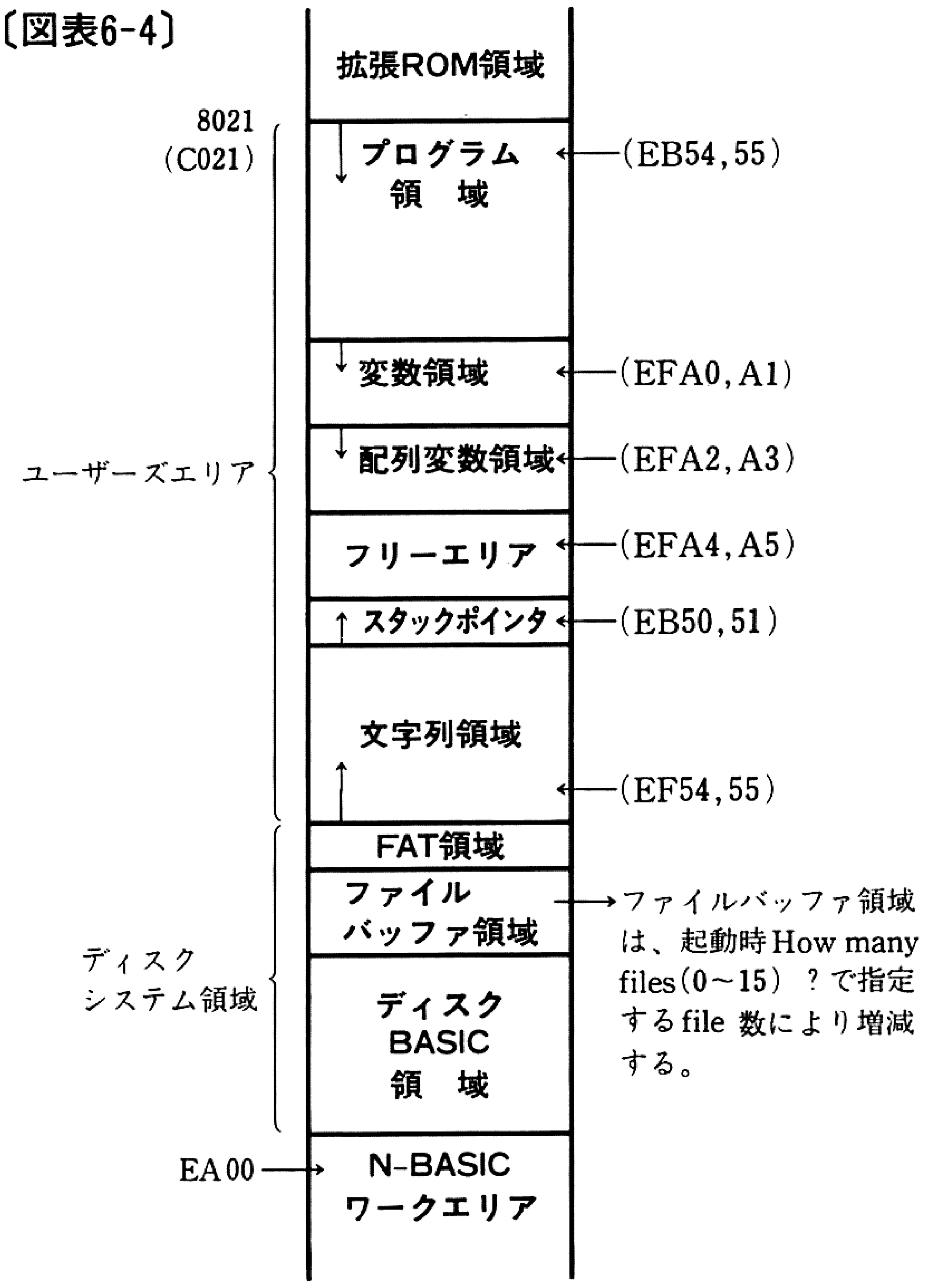
6-2.ディスク使用時のユーザーズエリア

Disk BASIC起動時には、Disk BASICインタプリタおよびそのシステム作業領域がEA00番地から始まるN-BASICのワークエリアの手前に設定され、6～10Kバイトを占有するので、ユーザーズエリアは、32Kシステムの場合17～21Kバイトに狭まります。（右図）

17～21Kバイトと幅があるのは、起動時に指定するファイル数によりファイルバッファ領域が増減するためです。

その詳細は次頁の図とアドレス表に示します。

〔図表6-4〕



ディスクシステムを起動すると、最初に画面に次のように表示されます。

```
Disk version [14-Nov-1980]
How many files(0-15)?
```

この時入力する数値により、ファイルバッファの数が決まります。例えば3と入力すれば＃0から＃3まで、各256バイトのバッファが確保されます。

数値を入力せずに RET キーのみを押すと、自動的に＃5までバッファが確保されます。

入力したファイルバッファ数は、E9D4番地に入りますので確認してみてください。

ファイル数を入力すると続いて次の様に表示されます。

```
Disk version [14-Nov-1980]
How many files(0-15)? 3
NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Microsoft

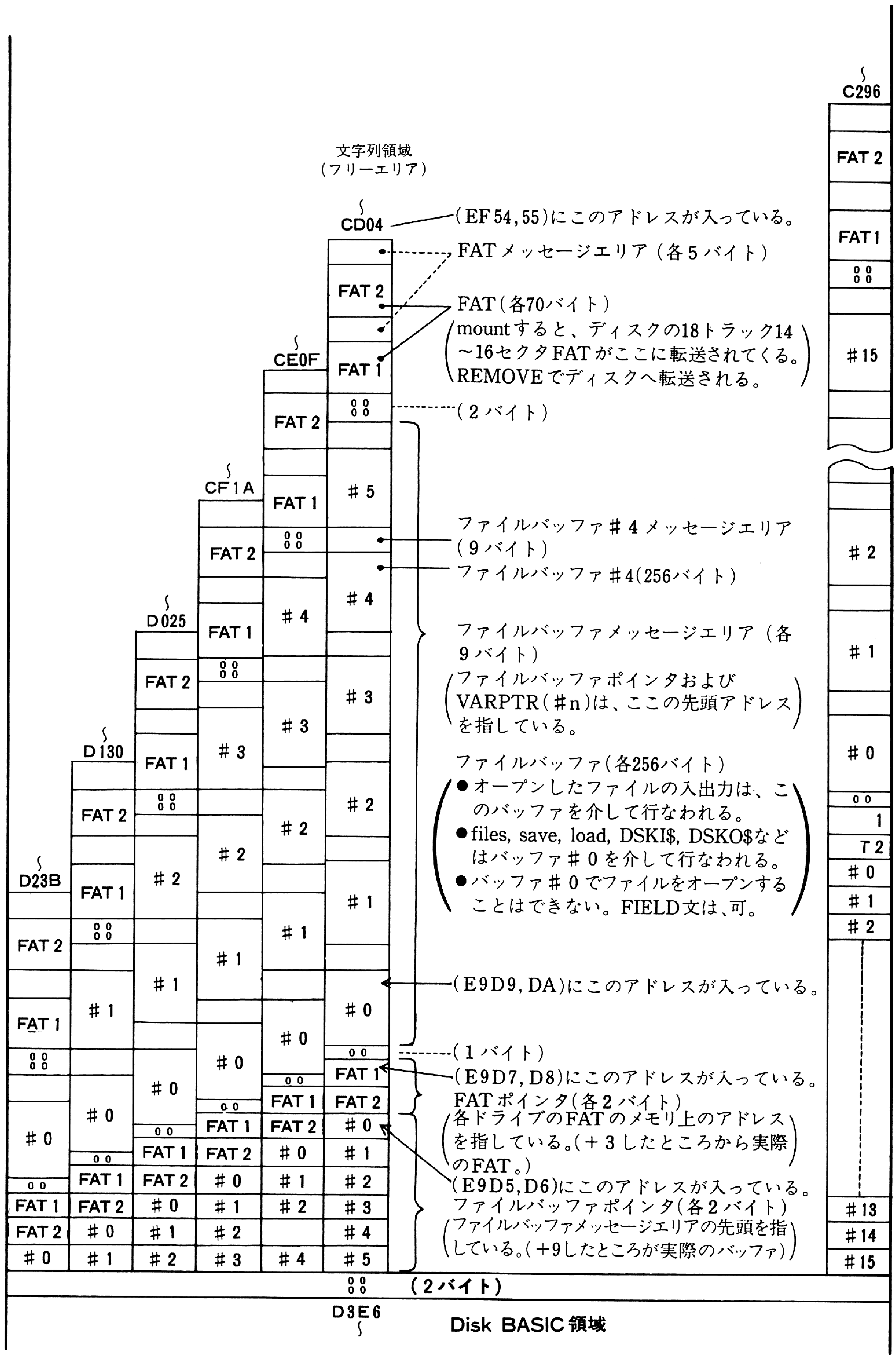
Ok
```

この本で記載する内容は、Disk version〔14-Nov-1980〕に関するもので、もし、起動時、次のように表示される旧バージョンの場合は、バッファ領域などの番地が若干異なるので、ご注意ください。

```
DISK VERSION
How many files(0-15)?
```

マイコンショップへ行けば、新バージョンのシステムをコピーしてもらえます。

〔図表6-5〕バッファ領域図



〔図表6-6〕バッファエリア アドレス一覧表

オープンファイル数		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
フリーエリア最終アドレス		D238	D130	D025	CF1A	CE0F	CD04	CBF9	CAEE	C9E3	C8D8	C7CD	C6C2	C5B7	C4AC	C3A1	C296
ドライブ2FAT		D241 D286	D136 D17B	D02B D070	CF20 CF65	CE15 CE5A	CD0A CD4F	CBFF CC44	CAF4 CB39	C9E9 CA2E	C8DE C923	C7D3 C818	C6C8 C70D	C5BD C602	C4B2 C4F7	C3A7 C3EC	C29C C2E1
ドライブ1FAT		D28C D2D1	D181 D1C6	D076 D0BB	CF6B CF80	CE60 CEA5	CD55 CD9A	CC4A CC8F	CB3F CB84	CA34 CA79	C929 C96E	C81E C863	C713 C758	C608 C640	C4FD C542	C3F2 C437	C2E7 C32C
＃15	ポインタ																C32F
	エリア																C338 C437
＃14	ポインタ															C43A	C438
	エリア															C443 C542	C441 C540
＃13	ポインタ														C545	C543	C541
	エリア														C54E C64D	C54C C64B	C54A C649
＃12	ポインタ													C650	C64E	C64C	C64A
	エリア													C659 C758	C657 C756	C655 C754	C653 C752
＃11	ポインタ												C758	C759	C757	C755	C753
	エリア												C764 C863	C762 C861	C760 C85F	C75E C85D	C75C C85B
＃10	ポインタ											C866	C864	C862	C860	C85E	C85C
	エリア											C86F C96E	C86D C96C	C86B C96A	C869 C968	C867 C966	C865 C964
＃ 9	ポインタ										C971	C96F	C96D	C96B	C969	C967	C965
	エリア										C97A CA79	C978 CA77	C976 CA75	C974 CA73	C972 CA71	C970 CA6F	C96E CA6D
＃ 8	ポインタ									CA7C	CA7A	CA78	CA76	CA74	CA72	CA70	CA6E
	エリア									CA85 CB84	CA83 CB82	CA81 CB80	CA7F CB7E	CA7D CB7C	CA7B CB7A	CA79 CB78	CA77 CB76
＃ 7	ポインタ								CB87	CB85	CB83	CB81	CB7F	CB7D	CB7B	CB79	CB77
	エリア								CB90 CC8F	CB8E CC8D	CB8C CC8B	CB8A CC89	CB88 CC87	CB86 CC85	CB84 CC83	CB82 CC81	CB80 CC7F
＃ 6	ポインタ							CC92	CC90	CC8E	CC8C	CC8A	CC88	CC86	CC84	CC82	CC80
	エリア							CC9B CD9A	CC99 CD98	CC97 CD96	CC95 CD94	CC93 CD92	CC91 CD90	CC8F CD8E	CC8D CD8C	CC8B CD8A	CC89 CD88
＃ 5	ポインタ						CD9D	CD9B	CD99	CD97	CD95	CD93	CD91	CD8F	CD8D	CD8B	CD89
	エリア						CDA6 CEA5	CDA4 CEA3	CDA2 CEA1	CDA0 CE9F	CD9E CE9D	CD9C CE9B	CD9A CE99	CD98 CE97	CD96 CE95	CD94 CE93	CD92 CE91
＃ 4	ポインタ					CEA8	CEA6	CEA4	CEA2	CEA0	CE9E	CE9C	CE9A	CE98	CE96	CE94	CE92
	エリア					CEB1 CFB0	CEAF CFAE	CEAD CFAC	CEAB CFAA	CEA9 CFA8	CEA7 CFA6	CEA5 CFA4	CEA3 CFA2	CEA1 CFA0	CE9F CF9E	CE9D CF9C	CE9B CF9A
＃ 3	ポインタ				CFB3	CFB1	CFAF	CFAD	CFAB	CFA9	CFA7	CFA5	CFA3	CFA1	CF9F	CF9D	CF9B
	エリア				CFBC D0BB	CFBA D0B9	CFB8 D0B7	CFB6 D0B5	CFB4 D0B3	CFB2 D0B1	CFB0 D0AF	CFAE D0AD	CFAC D0AB	CFAA D0A9	CFA8 D0A7	CFA6 D0A5	CFA4 D0A3
＃ 2	ポインタ			D0BE	D0BC	D0BA	D0B8	D0B6	D0B4	D0B2	D0B0	D0AE	D0AC	D0AA	D0A8	D0A6	D0A4
	エリア			D0C7 D1C6	D0C5 D1C4	D0C3 D1C2	D0C1 D1C0	D0BF D1BE	D0BD D1BC	D0BB D1BA	D0B9 D1B8	D0B7 D1B6	D0B5 D1B4	D0B3 D1B2	D0B1 D1B0	D0AF D1AE	D0AD D1AC
＃ 1	ポインタ		D1C9	D1C7	D1C5	D1C3	D1C1	D1BF	D1BD	D1BB	D1B9	D1B7	D1B5	D1B3	D1B1	D1AF	D1AD
	エリア		D1D2 D2D1	D1D0 D2CF	D1CE D2CD	D1CC D2CB	D1CA D2C9	D1C8 D2C7	D1C6 D2C5	D1C4 D2C3	D1C2 D2C1	D1C0 D2BF	D1BE D2BD	D1BC D2BB	D1BA D2B9	D1B8 D2B7	D1B6 D2B5
＃ 0	ポインタ	D2D4	D2D2	D2D0	D2CE	D2CC	D2CA	D2C8	D2C6	D2C4	D2C2	D2C0	D2BE	D2BC	D2BA	D2B8	D2B6
	エリア	D2DD D3DC	D2DB D3DA	D2D9 D3D8	D2D7 D3D6	D2D5 D3D4	D2D3 D3D2	D2D1 D3D0	D2CF D3CE	D2CD D3CC	D2CB D3CA	D2C9 D3C8	D2C7 D3C6	D2C5 D3C4	D2C3 D3C2	D2C1 D3C0	D2BF D3BE
FATポインタ		D3DE D3E1	D3DC D3DF	D3DA D3DD	D3D8 D3DB	D3D6 D3D9	D3D4 D3D7	D3D2 D3D5	D3D0 D3D3	D3CE D3D1	D3CC D3CF	D3CA D3CD	D3C8 D3CB	D3C6 D3C9	D3C4 D3C7	D3C2 D3C5	D3C0 D3C3
ファイルバッファ ポインタ		D3E2 D3E3	D3E0 D3E3	D3DE D3E3	D3DC D3E3	D3DA D3E3	D3D8 D3E3	D3D6 D3E3	D3D4 D3E3	D3D2 D3E3	D3D0 D3E3	D3CE D3E3	D3CC D3E3	D3CA D3E3	D3C8 D3E3	D3C6 D3E3	D3C4 D3E3
ディスクBasic先頭アドレス		D3E6															

〔図表6-7〕ファイルバッファ メッセージエリア（9バイト）

BUFFER AD（ポインタの指す アドレス）	ファイルモード	00…CLOSE 04…ランダム アクセス 01…FOR INPUT 08…FOR APPEND 02…FOR OUTPUT VARPTR（#n）関数やファイルバッファポインタは、このアドレスを指している。
AD+1	先頭クラスタ	Open したFileが格納されている先頭のクラスタ番号。
AD+2	アクセス中の クラスタ数	現在アクセス中のクラスタの番号。
AD+3	アクセス中の セクタ数	〃
AD+4	ドライブ番号	ファイルがオープンされたドライブ番号－1
AD+5	バッファー カウント	ファイルの何文字までデータがあるか。 最後まであるとき00
AD+6	カレントバッファーカウント	バッファの何文字目まで書き込まれているかを示す。
AD+7	ファイル属性	00…Close 18…ライトプロテクト 08…属性なし 48…リードアフターライト
AD+8	レコード中の桁数（左から何桁目）	

〔図表6-8〕FAT メッセージエリア（5バイト）

AD－2	リムーブフラッグ	メモリー上のFATの書きかえがあったかどうかを示す。
AD－1	属 性	00…属性なし 10…ライトプロテクト 40…リードアフターライト
FAT AD	アクセス トラック	最後にアクセスしたトラックNo. REMOVEすると〝FF〞になる。
AD+1		
AD+2	未使用 クラスタ数	ディスケットの未使用のクラスタ数。

6-3.ディスク用ダンプリスト

ディスクの中味を見ようとする場合、先ずDSKI$コマンドでドライブNo. Track No. Sector No. を指定して実行すると、ディスクバッファ＃0に、指定したセクタの内容(256バイト）が転送されてきます。そこで1-12のダンプリストプログラムを一部書き変えて、ディスク用ダンプリストプログラムとしたものを次に示します。

```
100 WIDTH 80,25:CONSOLE 0,25,0:CLEAR 1000:DEFINT A-Z:AD=VARPTR(#0)+9
110 INPUT "Drive  No.      ";D
120 T$="":INPUT "Track  No.      ";T$:IF T$="" THEN 140 ELSE T=VAL(T$)
130 INPUT "Sector No.      ";S:PRINT
140 DY$=DSKI$(D,T,S):C$=RIGHT$("0"+HEX$(T*2+(S-1)¥8),2):I=AD
150 PRINT USING "Drive #   Track ##   Sector ##   Cluster &&";D,T,S,C$
160 G=0:FOR J=0 TO 15:K=PEEK(I+J):PRINT RIGHT$("0"+HEX$(K),2)" ";
170 G=G+K:NEXT:PRINT " "RIGHT$("00"+HEX$(G),3)"   ";
180 FOR J=0 TO 15:K=PEEK(I+J)
190 IF K<32 OR K>247 THEN PRINT "・"; ELSE PRINT CHR$(K);
200 NEXT:PRINT:I=I+16:IF I-AD<256 THEN 160
210 PRINT:S=S+1:IF S>16 THEN T=T+1:S=1
220 GOTO 120
```

このプログラムを実行して、ドライブNo. 1または2（拡張ドライブ使用の場合は、1～4）、トラックNo. 0～34、セクタNo. 1～16を入力すれば、指定のセクタが表示されます。続くセクタを表示させたいときは、Track.No. ?□で RET のみを押してください。

プリンタに出力する場合、160行以降のprint文(7ヶ所)を、lprintに変更するか、画面プリントの項を参照してください。

それでは、実際に実行して、IPLの部分を見てみましょう。

```
run
Drive  No.      ? 1
Track  No.      ? 0
Sector No.      ? 1

Drive 1   Track  0   Sector  1   Cluster 00
C3 32 C0 F5 3E 01 B7 CD 99 00 38 1D 21 00 01 19  596  テ2ﾀ時>・ｷﾍｺ・8・!・・・
E5 11 00 EA AF 93 5F 3E 00 9A 57 19 E1 38 07 EB  6D4  ◣・・◆ｯ├_>・W・├8・♣
0C 79 FE 11 38 DE F1 B7 C9 F1 37 C9 AF 32 C7 ED  9A1  ・y・・8ﾞ円ｷﾉ円7ﾉｯ2ｽO
37 C9 11 00 C1 01 02 00 AF 32 C5 ED CD 03 C0 38  630  7ﾉ・・ﾁ・・・ｯ2ﾅOﾍ・ﾀ8
EB 04 0E 01 CD 03 C0 38 E3 04 0E 01 CD 03 C0 38  584  ♣・・・ﾍ・ﾀ8ｮ・・・ﾍ・ﾀ8
DB 21 FE FF 22 52 EB 3E 00 32 4D EB 32 98 ED 32  7E9  ロ!・・"R♣>・2M♣2┌O2
46 EB 21 58 EF 22 56 EF 21 C2 EF 22 2A F0 21 03  732  F♣!X\"V\!ﾂ\"*X!・
C1 CD ED 52 CD 5C 3E 38 F5 D7 B7 3E 05 28 0E CD  835  ﾁﾍORﾍ¥>8時ﾗｷ>・(・ﾍ
D5 4C 20 EA 7A B7 20 E6 7B FE 10 30 E1 32 D4 E9  8EB  ﾕL ◆zｷ ◥{・・0├2ﾔ♥
3C 87 2F 3C 5F 16 FF 21 E4 D3 19 22 D5 E9 E5 3A  792  <■/<_・・!◢ﾓ・"ﾕ♥◣:
C7 ED 87 2F 3C 5F 19 22 D7 E9 2B EB E1 01 09 01  702  ｽO■/<_・"ﾗ♥+♣├・・・
AF 91 4F F5 3E 00 98 47 00 3A D4 E9 3C 00 00 00  5D4  ｯ┬O時>・┌G・:ﾔ♥<・・・
00 00 00 00 00 EB 09 EB 73 23 72 23 3D 20 F6 F1  54E  ・・・・・♣・♣s#r#= 分円
B7 01 B5 FF 3E 00 F5 3A C7 ED 2A D7 E9 20 E6 1B  898  ｷ・ｵ・>・時:ｽO*ﾗ♥ ◥・
1B F1 D5 2A D5 E9 5E 23 56 21 09 00 19 22 D9 E9  6C7  ・円ﾕ*ﾕ♥^#V!・・・"ﾙ♥
AF 3D 32 F6 E9 CD 56 E2 AF 32 F6 E9 CD 7F D5 E1  AC4  ｯ=2分♥ﾍVｷｯ2分♥ﾍ ﾕ├
```

この内容と、C000番地からの内容を比べて見てください。同じものが入っていることがわかります。

他の部分も調べてみてください。

存在しないドライブNo. トラックNo. セクタNo. を指定するとエラーになりますので、必要であれば防止するようにしてください。

6-4.ディレクトリ

ディレクトリは、何という名のファイルが、どのクラスタから、どういう属性で格納されているかを記録しておく部分で、住所録みたいな役目をします。

ディレクトリは、18トラックの第1セクタから第12セクタにあります。

ディスクダンプリストで見てみましょう。

〔図表6-9〕

```
Drive  No.        ? 1
Track  No.        ? 18
Sector No.        ? 1

Drive  1    Track  18    Sector   1    Cluster  24
52 45 46 4F 52 4D 20 20 20 80 23 FF FF FF FF FF  7C9  REFORM     _#·····
56 44 55 4D 50 20 20 20 20 C0 26 FF FF FF FF FF  7ED  VDUMP      ﾀ&·····
53 45 41 52 43 48 42 41 53 80 20 FF FF FF FF FF  827  SEARCHBAS_ ·····
44 49 52 64 6D 70 20 20 20 80 1F FF FF FF FF FF  81A  DIRdmp     _······
66 6F 72 6D 61 74 20 20 20 90 29 FF FF FF FF FF  89D  format     ┴)·····
62 61 63 6B 75 70 20 20 20 90 1E FF FF FF FF FF  87F  backup     ┴······
46 73 68 65 65 74 20 20 20 80 2A FF FF FF FF FF  864  Fsheet     _*·····
4D 45 4D 64 6D 70 20 20 20 80 1D FF FF FF FF FF  818  MEMdmp     _······
44 53 4B 64 6D 70 20 20 20 80 2B FF FF FF FF FF  829  DSKdmp     _+·····
45 72 72 6F 72 20 20 20 20 80 1C FF FF FF FF FF  821  Error      _······
4C 70 72 69 6E 74 4D 50 20 80 2C FF FF FF FF FF  8DD  LprintMP   _,·····
41 53 43 49 49 20 20 20 20 80 19 FF FF FF FF FF  77D  ASCII      _······
46 41 54 64 73 70 20 20 20 F0 2D FF FF FF FF FF  89A  FATdsp     X-·····
53 54 4D 63 70 79 20 20 20 80 1A FF FF FF FF FF  835  STMcpy     _······
44 2D 65 64 69 74 20 20 20 80 2E FF FF FF FF FF  820  D-edit     _······
53 65 63 72 65 74 20 20 20 80 1B FF FF FF FF FF  85C  Secret     _······
```

ファイル名(6バイト)　　拡張子(3バイト)　　(未使用(5バイト)

→ そのファイルが格納されている先頭のクラスタ番号(16進数0～45H)

先頭バイト
00…Killされたファイルを示す。
FF…未使用を示す。

属性

値	意味
00	アスキー形式
10	アスキー形式、書込み禁止
40	アスキー形式、リードアフタライト
80	非アスキー形式
90	非アスキー形式、書込み禁止
C0	非アスキー形式、リードアフタライト

注1) アスキーSave…Save"ファイル名",a

注2) 属性Set"ファイル名","P"…書込み禁止
Set"ファイル名","R"…リードアフタライト

6-5.ID

IDは、第18トラック、第13セクタの先頭の１バイトであり、残り255バイトは、未使用となっています。

IDは、ファイル全体の属性を示します。

書込みは、DSKO$コマンドを用いますのでMOUNTしておく必要はありません。

その例を次に示します。

〔図表6-10〕

```
list
100 FIELD #0, 1 AS ID$
110 LSET ID$=CHR$(&H10)
120 DSKO$ 1,18,13
OK

run
OK

mount 1
OK

save "Test"
File write protected
OK
```

値	意　　味	属性文字
0 0	属性なし	〝〟
1 0	書込み禁止	〝P〟
4 0	リードアフタライト	〝R〟

```
                     ↓ドライブ番号
print attr$ (1)
  P
OK
```

例のように、書込み禁止コード〝10 H〟を書き込むと、新しいプログラムをSAVEしようとしてもできません。

注）書き込み後、再度MOUNTしないと、本体内のFATの属性は変わらないので注意してください。

6-6.FAT

ファイルは、クラスタ(８セクタ)単位で管理されており、ディスケットあたり70（０～69）あるクラスタの使用状況が記録されている部分をFAT（File Allocation Table）と呼びます。

記録場所は、第18トラックの第14～16セクタで、各々70バイトづつ同じものが各セクタに書込まれています。残り各186バイトは未使用です。

6－3のディスク用ダンプリストで見てみましょう。

```
run
Drive  No.      ? 1
Track  No.      ? 18
Sector No.(S,E) ? 14,14

Drive 1   Track 18   Sector 14   Cluster 25
FE FE FE FE FE FE FF FF FF FF FF FF FF FF FF FF  FEA  ................
FF FF FF FF FF FF FF FF FF FF FF C5 C2 C3 C5 C4  EC8  ...........ﾅﾂﾃﾅﾄ
C8 C2 21 22 FE FE 27 28 C1 C4 C6 C3 C8 C2 FF FF  AAE  ﾈﾂ!"..'(ﾁﾄﾆﾃﾈﾂ..
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF 00 00 00 00 00 00 00 00 00 00  5FA  ................
00 00 00 00 00 00 00 00 [illegible] 00 00 00  000  ................
```

これでは少し分かりにくいので、トラック番号と、クラスタ番号と対応させて出力するプログラムと実行例を次に示します。

```
100 DIM FT(69):FIELD #0,70 AS FT$
110 INPUT "Drive No. ";D:DY$=DSKI$(D,18,14)
120 K=0:FOR I=0 TO 69:FT(I)=ASC(MID$(FT$,I+1,1)):IF FT(I)=255 THEN K=K+1
130 NEXT:PRINT CHR$(12);"*****  DRIVE";D;" FREE";K;" *****":PRINT
140 FOR I=0 TO 4:PRINT "No.TR CL FAT    ";:NEXT
150 FOR I=0 TO 69:LOCATE (I¥16)*16,(I MOD 16)+3
160 T=I¥2
170 C$=RIGHT$("0"+HEX$(I),2):FT$=RIGHT$("0"+HEX$(FT(I)),2)
180 PRINT USING "## ## &&  &&";I,T,C$,FT$
190 NEXT:LOCATE0,20
```

```
*****  DRIVE 1  FREE 45  *****

No.TR CL FAT    No.TR CL FAT    No.TR CL FAT    No.TR CL FAT    No.TR CL FAT
 0  0 00  FE    16  8 10  FF    32 16 20  C8    48 24 30  FF    64 32 40  FF
 1  0 01  FE    17  8 11  FF    33 16 21  C2    49 24 31  FF    65 32 41  FF
 2  1 02  FE    18  9 12  FF    34 17 22  21    50 25 32  FF    66 33 42  FF
 3  1 03  FE    19  9 13  FF    35 17 23  22    51 25 33  FF    67 33 43  FF
 4  2 04  FE    20 10 14  FF    36 18 24  FE    52 26 34  FF    68 34 44  FF
 5  2 05  FE    21 10 15  FF    37 18 25  FE    53 26 35  FF    69 34 45  FF
 6  3 06  FF    22 11 16  FF    38 19 26  27    54 27 36  FF
 7  3 07  FF    23 11 17  FF    39 19 27  28    55 27 37  FF
 8  4 08  FF    24 12 18  FF    40 20 28  C1    56 28 38  FF
 9  4 09  FF    25 12 19  FF    41 20 29  C4    57 28 39  FF
10  5 0A  FF    26 13 1A  FF    42 21 2A  C6    58 29 3A  FF
11  5 0B  FF    27 13 1B  C5    43 21 2B  C3    59 29 3B  FF
12  6 0C  FF    28 14 1C  C2    44 22 2C  C8    60 30 3C  FF
13  6 0D  FF    29 14 1D  C3    45 22 2D  C2    61 30 3D  FF
14  7 0E  FF    30 15 1E  C5    46 23 2E  FF    62 31 3E  FF
15  7 0F  FF    31 15 1F  C4    47 23 2F  FF    63 31 3F  FF

Ok
```

FATの各1バイトのデータは、各々次表に示すような意味をもっており、ファイルが、どこにどういう順序で入っているかを示しています。

〔図表6-11〕

FAT Data	意　　　味
0～45H	使用中のクラスタで、後続するクラスタを持つ、 数値は、続いて使用されているクラスタ番号を示す。
C1～C8	使用中のクラスタのうちの最後のクラスタを示す。 下位4ビット(1～8)が、そのクラスタで使用しているセクタ数を示す。
FE	予約済のクラスタ……ファイルとして、使うことはできない。 IPL, DISK BASIC PROGRAMの部分(0～5クラスタ)やディレクトリ ID・FATの部分(24～25クラスタ)がこれに相当する。
FF	未使用のクラスタ。

先に調べた、ディレクトリのダンプリストより、例えば、"REFORM"というプログラムの場合、格納先の先頭クラスタは、23Hであることがわかります。

FATの23Hの内容は、22Hであり、22Hクラスタに続いていることを意味します。更に22Hの内容は、21Hで、続いて21Hの内容は、C2Hであり、21Hクラスタを2セクタ使って終わっていることがわかります。

すなわち、"REFORM"ファイルは、23H、22H、21Hの順に2クラスタと2セクタ使用していることがわかります。

6-7.ファイルの Location

ディレクトリにファイル名と属性と、格納先の先頭クラスタ番号、また、FAT にファイルの格納場所および順序が記憶されているので、両方を参照すれば、ファイルの属性・格納場所がわかります。

次に、その応用プログラムと実行例を示します。

```
100 CLEAR 1000:DIM FT(69),F$(15),E$(15),AT$(15),CL$(15)
110 FIELD #0,70 AS FT$
120 FOR I=0 TO 15
130 FIELD #0,I*16 AS DY$,6 AS F$(I),3 AS E$(I),1 AS AT$(I),1 AS CL$(I):NEXT
140 INPUT "Drive No. ";D:DY$=DSKI$(D,18,14)
150 K=0:FOR I=0 TO 69:FT(I)=ASC(MID$(FT$,I+1,1)):IF FT(I)=255 THEN K=K+1
160 NEXT:PRINT CHR$(12);
170 PRINT "*****  DRIVE";D;"FREE";K;" *****":PRINT
180 PRINT "FILE-NAME ATTR LOCATION"
190 FOR S=1 TO 12:DY$=DSKI$(D,18,S):FOR F=0 TO 15
200 AT=ASC(AT$(F)):AT$="N":CL=ASC(CL$(F)):LC$=RIGHT$("0"+HEX$(CL),2)
210 IF AT<&H80 THEN AC$=" " ELSE AC$="."
220 IF AT=&H10 OR AT=&H90 THEN AT$="P"
230 IF AT=&H40 OR AT=&HC0 THEN AT$="R"
240 IF LEFT$(F$(F),1)=CHR$(  0)THEN 290
250 IF LEFT$(F$(F),1)=CHR$(255)THEN END
260 IF FT(CL)>&H45 THEN LC$=LC$+"("+CHR$(FT(CL)-&H90)+")":GOTO 280
270 LC$=LC$+" "+RIGHT$("0"+HEX$(FT(CL)),2):CL=FT(CL):GOTO 260
280 PRINT F$(F);AC$;E$(F);" ";AT$;"   ";LC$
290 NEXT F,S
```

実行すると、ドライブ番号を聞いてきますので調べたいDiskの入っているドライブ番号を入力すると、次に示すようにFile Locationが表示されます。

```
*****  DRIVE 1 FREE 45  *****

FILE-NAME ATTR LOCATION
REFORM.     N   23 22 21(2)
VDUMP  .    R   26 27 28(1)
SEARCH.BAS  N   20(8)
Files  .    N   1F(4)
format.     P   29(4)
backup.     P   1E(5)
Ftable.     N   2A(6)
ASCdmp.     N   1D(3)
DSKdmp      P   2B(3)
Error  .    N   1C(2)
PTRsub      N   2C(8)
V-list dat  R   1B(5)
FATdsp.     N   2D(2)
```

最後の(　)内の数値は、そのクラスタの第何セクタまで使用しているかを示します。
また、このプログラムは、ディスクがMOUNTされていなくても、実行できます。

6-8.ファイルの転送

ディスク中のファイルを、他のディスクにコピーするとき、プログラムファイルであれば、LOAD命令で一旦本体内メモリに入れ、SAVE命令で目的のディスクへ書込めばOKです。

```
load"ABC"
OK
save"2:ABC"
OK
```

ところがデータファイルの場合、LOAD命令で、本体内メモリへ読込むことができません。

```
load"2:Zaiko.dat"
Direct statement in file
OK
Communications Buffer Overflow
OK
```

データファイルの場合、次の様にそれぞれのファイルをOPENして、R＝1からLOFまでGETとPUTをくりかえす必要があります。

```
100 OPEN"Zaiko.dat"AS#1
110 OPEN"2:Zaiko.dat"AS#2
120 FIELD#1,128 AS A1$,128 AS B1$
130 FIELD#2,128 AS A2$,128 AS B2$
140 FOR R=1 TO LOF(1):GET #1,R
142 LSET A2$=A1$:LSET B2$=B1$
148 PUT #2,R:NEXT
150 REMOVE:END
```

ところが、この方法でプログラムファイルを転送すると、属性は、アスキー形式となってしまいますので、アスキー形式でSAVEされたプログラムであれば転送可能ですが、非アスキー形式でSAVEされたファイルですと、データファイルをLOADしたときと同じエラーメッセージが表示されLOADできません。即ちファイルの属性もコピーする必要があるわけです。

次に属性も含めてファイルを転送するプログラムを示します。
これですと、プログラムファイルであろうとデータファイルであろうと、関係なしに、ドライブ1から2へ、または、ドライブ2から1へと、自由にファイルの転送ができます。

```
100 CLEAR 1000:WIDTH80,25:DIM F$(15),E$(15),A$(15)
110 FOR I=0 TO 15:FIELD#0,I*16 AS DY$,6 AS F$(I),3 AS E$(I),1 AS A$(I):NEXT
120 FIELD#1,128 AS A1$,128 AS B1$
130 FIELD#2,128 AS A2$,128 AS B2$
140 PRINT "*** Drive 1 ***      Free";DSKF(1):FILES :PRINT
150 PRINT "*** Drive 2 ***      Free";DSKF(2):FILES2:PRINT
160 O$="":INPUT "Origin      File Name ";O$:IFO$=""THEN 390
170 INPUT "Destination File Name ";D$:ER=0
180 F$=O$:GOSUB 250:IF ER=1 THEN 160 ELSE GOSUB 340:IF ER=1 THEN 140
190 A$=A$(I):OPEN RIGHT$(STR$(D),1)+":"+F0$+"."+E0$ AS #1
200 F$=D$:GOSUB 250:IF ER=1 THEN 170
210 OPEN RIGHT$(STR$(D),1)+":"+F0$+"."+E0$ AS #2
220 FOR I=1 TO LOF(1):GET #1,I:LSET A2$=A1$:LSET B2$=B1$: PUT #2,I:NEXT
230 GOSUB 340:LSET A$(I)=A$:DSKO$ D,18,S
240 CLOSE:PRINT CHR$(12);:GOTO140
250 '      File Name Reform Sub
260 C=INSTR(F$,":"):IF C=0 THEN D=1:GOTO 280
270 D=VAL(LEFT$(F$,C-1)):F$=MID$(F$,C+1)
280 IF D<1 OR D>2 THEN PRINT "Bad Drive Number !":BEEP:ER=1:RETURN
290 P=INSTR(F$,"."):IF P<>0 THEN 310
300 F0$=LEFT$(F$+SPACE$(6),6):E0$=MID$(F$+SPACE$(9),7,3):RETURN
310 F0$=LEFT$(F$,P-1):IFLEN(F0$)>6THENPRINT"Bad File Name !":BEEP:ER=1:RETURN
320 F0$=LEFT$(F0$+SPACE$(6),6):E0$=MID$(F$+SPACE$(3),P+1,3)
330 RETURN
340 '      File Search Sub
350 FOR S=1 TO 12:DY$=DSKI$(D,18,S):FOR I=0 TO 15
360 IF F0$=F$(I) AND E0$=E$(I) THEN RETURN
370 NEXT I,S
380 PRINT "File not Exist !":BEEP:ER=1:RETURN
390 PRINT:PRINT"remove":REMOVE
```

6-9.データディスクをシステムディスクに

データディスクとして使用していたディスクを中のプログラムやデータを壊さずにシステムディスク化したいとき、購入時システムディスク内に入っている〝back up〟プログラムは使えません。システムディスク化する基本的手順としては、

(1)データディスクの0～2トラックにデータが入っていないか確認する。
　(FATの第0～5クラスタがFFかどうか)

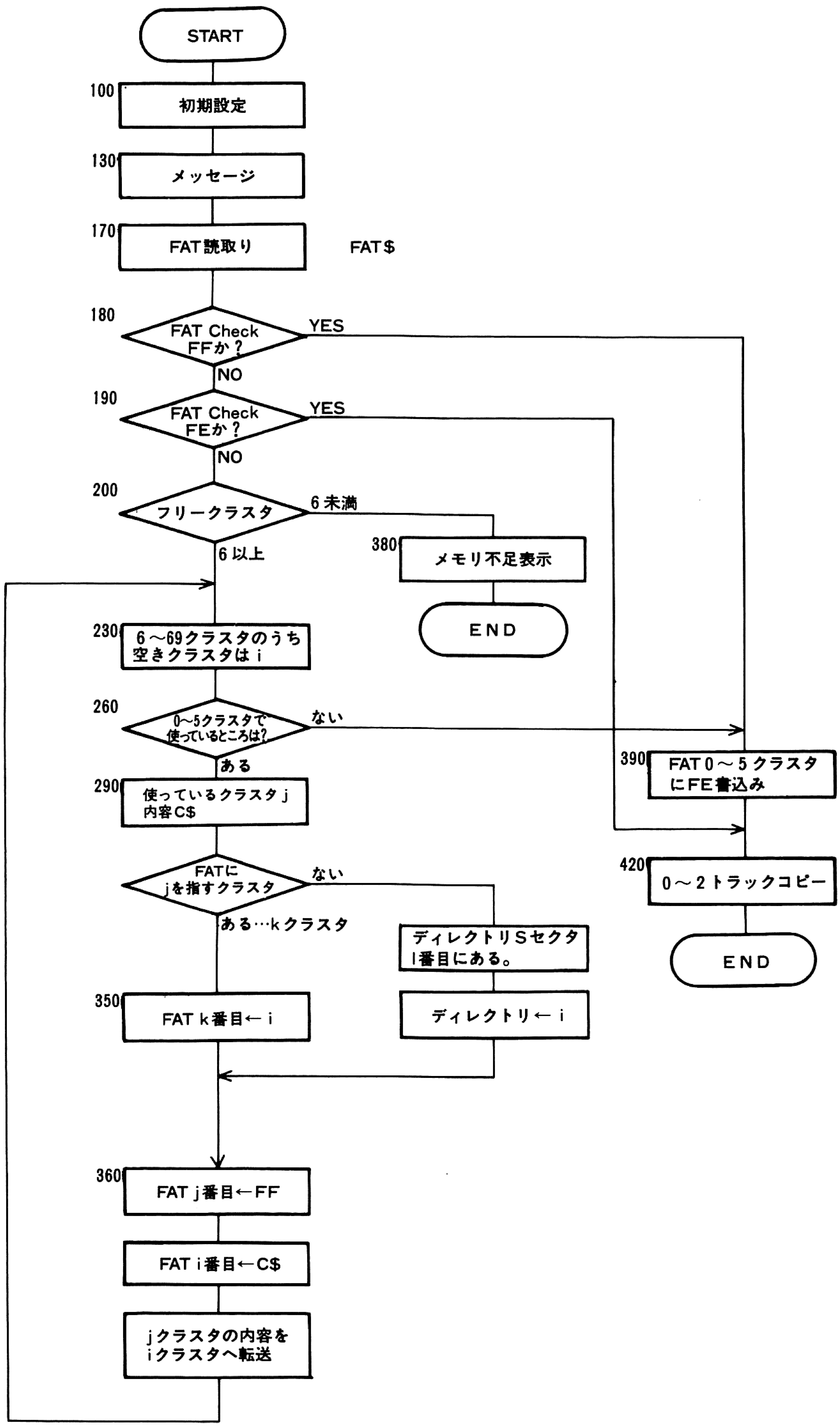
(2)システムディスクの0～2トラックの内容をデータディスクへ転送する。

(3)FAT(18トラック、14～16セクタ)の0～5クラスタ部を〝FE〟にする。

ですが、もし、データディスクの0～2トラック(Cluster 0～5)にデータがある場合には、3～34トラックの空いている部分にこのデータを転送するとともに、それに対応するFATも書き替える必要があります。

次に、そのフローチャートとプログラム例を示します。

〔図表6-12〕


START
100 初期設定
130 メッセージ
170 FAT読取り
FAT$
180 FAT Check FFか？
YES
NO
190 FAT Check FEか？
YES
NO
200 フリークラスタ
6未満
6以上
380 メモリ不足表示
END
230 6～69クラスタのうち空きクラスタはi
260 0～5クラスタで使っているところは?
ない
ある
290 使っているクラスタj 内容C$
FATに jを指すクラスタ
ない
ある…kクラスタ
ディレクトリSセクタ I番目にある。
350 FAT k番目←i
ディレクトリ←i
360 FAT j番目←FF
FAT i番目←C$
jクラスタの内容を iクラスタへ転送
390 FAT 0～5クラスタにFE書込み
420 0～2トラックコピー
END

プログラムをロード後、ドライブ1にシステムディスク、ドライブ2にデータディスクを入れて実行します。すべてDSKI$, DSKO$で読み書きするのでMOUNTする必要はありません。

```
100 REMOVE:CLEAR 1000:DIM S$(15),C$(15)
110 FIELD#0,70AS F$
120 FOR I=0 TO15:FIELD#0, I*16AS DY$,1AS S$(I),9AS DY$,1AS C$(15):NEXT
130 PRINT "Drive 1 ... System Disk"
140 PRINT "Drive 2 ... Data   Disk    Ready (y/n) ? ";
150 KY$=INPUT$(1):IF KY$<>"y" THEN 150
160 COLOR 6:PRINT:PRINT "Now Working ...":COLOR 0
170 DY$=DSKI$(2,18,14):FAT$=F$
180 IF LEFT$(FAT$,6)=STRING$(6,CHR$(255)) THEN 390
190 IF LEFT$(FAT$,6)=STRING$(6,CHR$(254)) THEN 420
200 N=0:FOR I=0 TO 69:IF MID$(FAT$,I+1,1)=CHR$(255) THEN N=N+1
210 NEXT
220 IF N<6 THEN 380
230 FOR I=6 TO 69:IF MID$(FAT$,I+1,1)=CHR$(255) THEN 250
240 NEXT I:GOTO 390
250 FOR J=0 TO 5:C$=MID$(FAT$,J+1,1):IF C$<CHR$(255) THEN 270
260 NEXT J:GOTO 390
270 FOR K=0 TO 69:IF MID$(FAT$,K+1,1)=CHR$(J) THEN 330
280 NEXT K
290 FOR S=1 TO 12:DY$=DSKI$(2,18,S)
300 FOR L=0 TO 15:IF S$(L)=CHR$(J) THEN 320
310 NEXT L,S:GOTO 370
320 LSET C$(L)=CHR$(I):DSKO$ 2,18,S:GOTO 340
330 MID$(FAT$,K+1,1)=CHR$(I)
340 MID$(FAT$,J+1,1)=CHR$(255):MID$(FAT$,I+1,1)=C$
350 FOR S=1 TO 8:DY$=DSKI$(2,J¥2,S+(JMOD2)*8)
360 DSKO$ 2,I¥2,S+(IMOD2)*8:NEXT S:GOTO 230
370 PRINT "FAT construction is bad.":BEEP:END
380 PRINT "Disk full ... Free Cluster";N:BEEP:END
390 MID$(FAT$,1,6)=STRING$(6,CHR$(254))
400 DY$=DSKI$(2,18,14):LSET F$=FAT$
410 FOR S=14 TO 16:DSKO$ 2,18,S:NEXT
420 FOR T=0 TO 2:FOR S=1 TO 16:DY$=DSKI$(1,T,S):DSKO$ 2,T,S:NEXT S,T
430 PRINT:PRINT "Complete"
```

6-10. Disk BASICの回復法・切離し法

Disk BASIC使用中に、何らかの理由で

1. ディスクユニットの電源を切った。
2. ホットリセットをかけた。
   (ストップキーを押しながらリセットボタンを押した。)

このようなとき、例えばMOUNTなどを実行しても、正常な動作をしません。

本体メモリ中のプログラムを消さずに、回復させるには、Basicインタプリタ中の00D2番地(Disk Initialize)をCALLすればOKです。

BASICでCALLするには、USR文を定義して実行できますが、面倒ならば、モニタモード

で上記番地へジャンプしても良いでしょう。この場合は、"Syntax Error"を表示してBASICコマンドレベルに戻ります。

```
① defusr=&hd2:u=usr(0)
   Ok
```

```
   mon
② *GD2
   Syntax error
   Ok
```

なお、ディスクがMOUNTされていた状態であったとしても、そのまま残っていますからFILES, REMOVE 等も問題なく使えます。もちろん、本体メモリ上のプログラムが消えたりするようなことはありません。

もし、ただ単にホットリセットしただけならば、ディスク用I/Oポートがリセットされただけなので、OUT命令でこのポート(コントロールポートアドレスFFH)にIC(μPD8255)の初期設定コードである91Hを出力してやれば、再びDisk BASICが、使用可能となります。

```
out &hff,&h91
Ok
```

また、ディスクユニットが接続してあるシステムで、Disk BASICを使用中、普通のN-BASICだけを使いたいときはどうすれば良いでしょうか？

ディスクユニットの電源を切ってリセットするのも1つの方法ですが、もっと簡単な方法があります。それは、[ESC]キーを押しながらRESETボタンを押すESC-RESET です。システムによって違いますが、数秒から十数秒後に、ディスクユニットなしで、リセットした状態になります。

6-11.ファイルのLOAD方法

通常のファイルのloadは、

load"ファイル名"(後の〔"〕は、省略しても良い)

で行ないますが、カタカナのファイル名などキーインするのが面倒な場合があります。このような場合、次の方法でLOADまたはRUNすると便利です。

1. 白黒80文字モード以外(72,40,36文字モードまたはカラー80文字モード)でfiles を実行します(ドライブ2の場合はfiles 2)。

```
files
REFORM.      3     VDUMP  .     3
SEARCH.BAS   1     Files  .     1
format.      1     backup.      1
Fsheet.      1     ASCdmp.      1
DSKdmp       1     Error  .     1
PTRsub       1     V-list dat   1
FATdsp.      1
OK
```

2. LOADまたは、RUNしたいファイル名のあるところまでカーソルを上げ、ファイル名の前に load〃または、 run〃をキーインし、[RET]キーを押します（ドライブ2の場合は、load〃2:またはrun〃2:)。
行の始めにあるときは、[INS]キーで上記のコマンドをキーインするスペースをあけ、それ以外は、スペースキーでカーソルを移動させキーインします。loadやrunの位置は、ファイル名の前であればどこでもかまいませんが、[〃]や[〃2:]は、ファイル名の直前でなければなりません。

例①

```
files
REFORM.      3     VDUMP  .     3
SEARCH.BAS   1     Files  .     1
format.      1     backup.      1
Fsheet.      1     ASCdmp.      1
load    "DSKdmp       1      Error  .     1
PTRsub       1     V-list dat   1
FATdsp.      1
OK
```

②

```
files
REFORM.      3     VDUMP  .     3
SEARCH.BAS   1     Files  .     1
format.      1     backup.      1
Fsheet.      1     ASCdmp.      1
load              "Error  .     1
PTRsub       1     V-list dat   1
FATdsp.      1
OK
```

③

```
files2
REFORM.      3     VDUMP  .     3
SEARCH.BAS   1     Files  .     1
format.      1     backup.      1
Fsheet.      1     ASCdmp.      1
load[        ]   "2:Error  .    1
PTRsub       1     V-list dat   1
FATdsp.      1
OK
```

6-12.ディスク内データの直接修正

ディスク内の任意の箇所を直接書き換えたい場合がよくあります。例えば、

1. 新しいプログラムやデータをSAVEやPUTしたが、最後にREMOVEしなかったとき。……FATが古いままになっている。
2. MOUNTしている状態で、別のディスクを入れて、REMOVEしてしまった。……FATが前に入れておいたディスクのものに書き換えられてしまっている。少々めんどうだがディレクトリは、元のままで、先頭クラスタはわかるので、ディスクダンプリストで各クラスタを調べて、FATを組立てて書換える。
3. IDをセットしたいがめんどう。
   18トラック第13セクタの1バイト目を書き換えれば良い。
4. IPLやシステムプログラムを一部変更したい。
5. 必要なプログラムをKILLしてしまった。…ディレクトリとFATを書き換える。

```
00 42 43 44 20 20 ·············· 3E ··············· B C D ···············
↑
└── 元のFile名が"ABCD"であったらA
    のアスキーコード41に書き換える。
```

先頭クラスタ番号が3Eであるので FAT(14～16セクタ)の3E番目の部分および、それに続いて使用しているクラスタがあれば対応するFATのFFを0～45H又は、C1～C8に書き換える。

6. データファイルの一部を修正したいがデータ変更用のプログラムを作っていない。または、変更プログラムを実行するのがめんどう。

などなど、列挙すればきりがありません。

このようなとき次に示すプログラムを実行すれば任意のトラック1セクタのデータが、画面上で次々に修正できます。

ディスクメモリの修正プログラム

```
100 WIDTH 80,25:CONSOLE 0,25,0:CLEAR 1000:DEFINT A-Z:AD=VARPTR(#0)+9
110 INPUT "Drive  No.       ";D
120 INPUT "Track  No.       ";T
130 INPUT "Sector No.       ";S:PRINT CHR$(12);
140 DY$=DSKI$(D,T,S):C$=RIGHT$("0"+HEX$(T*2+(S-1)¥8),2):I=AD
150 PRINT USING "Drive #   Track ##   Sector ##   Cluster &&";D,T,S,C$:PRINT
160 G=0:FOR J=0 TO 15:K=PEEK(I+J):PRINT RIGHT$("0"+HEX$(K),2)" ";
170 G=G+K:NEXT:PRINT " "RIGHT$("00"+HEX$(G),3)"  ";
180 FOR J=0 TO 15:K=PEEK(I+J)
190 IF K<32 OR K>247 THEN PRINT "･"; ELSE PRINT CHR$(K);
200 NEXT:PRINT:I=I+16:IF I-AD<256 THEN 160
210 '     *****  Edit  *****
220 N=0:M=0
230 X=(NMOD16)*3:Y=N¥16+2
240 LOCATE X,Y:K$=INPUT$(1):IF M=1 THEN 300
250 IF K$=CHR$(13) THEN 390
260 IF K$=CHR$(28) THEN N=N+ 1:IF N>255 THEN N=0
```

```
270 IF K$=CHR$(29) THEN N=N- 1:IF N<0   THEN N=255
280 IF K$=CHR$(30) THEN N=N-16:IF N<0   THEN N=N+256
290 IF K$=CHR$(31) THEN N=N+16:IF N>255 THEN N=N-256
300 IF K$>="0" AND K$=<"9" THEN 330
310 IF K$>="a" AND K$=<"f" THEN K$=CHR$(ASC(K$)-32):GOTO 330
320 ON M+1 GOTO 230,240
330 LOCATE X,Y:PRINT K$:IF M=0 THEN D$=K$:X=X+1:M=1:GOTO 240
340 M=0:D$=D$+K$:K=VAL("&h"+D$):POKE AD+N,K
350 G=0:FOR I=0 TO 15:G=G+PEEK(AD+(N¥16)*16+I):NEXT
360 LOCATE 49,Y:PRINT RIGHT$("00"+HEX$(G),3)
370 LOCATE 54+NMOD16,Y:IF K<32 OR K>247 THEN PRINT "・" ELSE PRINT CHR$(K)
380 N=N+1:GOTO 230
390 DSKO$ D,T,S:LOCATE 0,19:GOTO 120
```

使い方

1. runすると、ドライブNo. トラックNo. セクタNo. を聞いてきますので、各々10進法で入力します。
2. 指定したセクタの全256バイトが、Check sum, アスキーコードと共に画面に表示され、第1バイト(左上)の部分でカーソルが点滅します。
3. カーソルキー(⇄ ⇅ Shift)を使って修正したい位置にカーソルを持っていきます。
4. データ(0～9, a～f)をキーインします。a～fは小文字で入力してください。自動的に大文字になって表示されます。
5. 修正が終わって、RET キーを押すと、直ちにその内容が、ディスクへ書き込まれ次に修正すべきトラックNo. とセクタNo. を聞いてきます。
   RET を押す前に STOP キーでブレークすると、そのセクタのデータは、すべて元のままです。

6-13.シークレット ファイル

ファイルを保護するものにディスクやファイルの属性を指定する方法やプロテクトシールを貼る方法などがありますが、これらは、誰でも簡単に解除することができます。

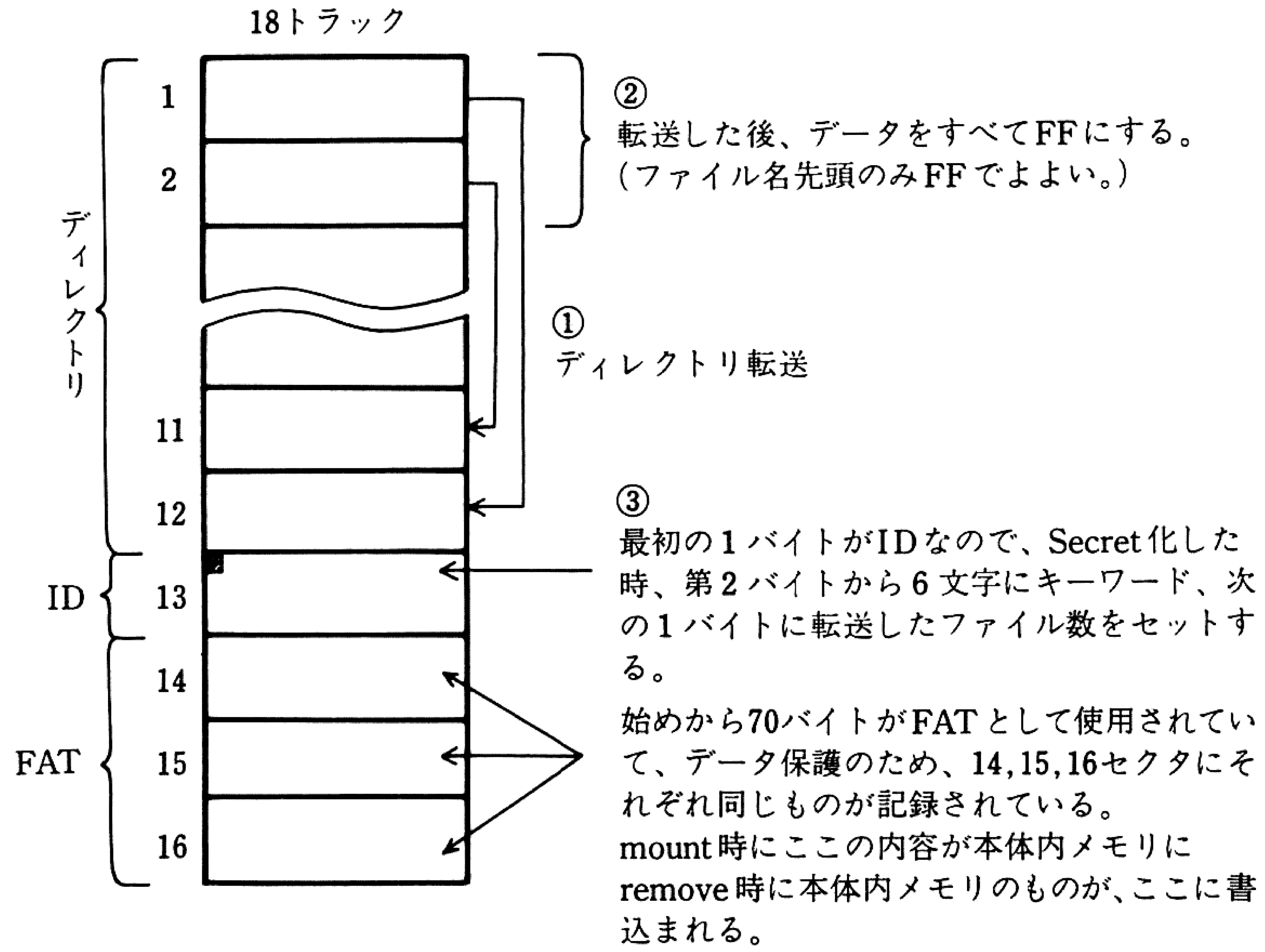
ここでは、1つのプロテクト方法について説明しますので、参考にしてください。

1. 18トラック第1セクタより始まるディレクトリを第12, 11, …セクタへと後方に転送する。
2. 移動した後のディレクトリをすべてFFにする。

というのがこの基本的な方法で、回復させるときは、その後SAVEしたファイルのディレクトリに続いて、退避させていたディレクトリを順次書込むというものです。

ディレクトリの少なくともファイル名の先頭をFFにすると、それ以降にディレクトリはないと判断し、FF以降から新たにSAVEするファイルのディレクトリが記録されていきます。

〔図表6-13〕



1. FILESを実行しても全く画面に表示されない。
2. すべてのファイルは、LOADできなくなります。
3. 新たなプログラムをSAVEすることは、通常通り可能です。
   このときFILESを実行すると、SECRET化後、SAVE又はOPENしたFileのみが表示されます。
4. Recover後は、新たにSAVEしたファイルと、SECRET化していたfileすべてがFILESにより表示されます。

次に、そのプログラムと実行例を示します。

```
100 CLEAR 2000:DIM F$(15)
110 FIELD#0,1 AS ID$,6 AS K$,1 AS N$
120 FOR I=0 TO 15:FIELD#0,I*16 AS DY$,16 AS F$(I):NEXT
130 INPUT"Drive No. ";D:REMOVE D
140 DY$=DSKI$(D,18,13)
150 IF K$="SECRET" THEN 290
160 '      *****  Secret  *****
170 PRINT "Do you make Secret File ? (y/n) ";:KY$=INPUT$(1)
180 PRINT:IF KY$<>"y" THEN END
190 S=1
200 DY$=DSKI$(D,18,S)
210 IF ASC(F$(0))=255 THEN 250 ELSE DSKO$ D,18,13-S
220 FOR I=0 TO 15:IF ASC(F$(I))=255 THEN DSKO$ D,18,S:GOTO 250
230 LSET F$(I)=STRING$(16,CHR$(255)):NEXT
240 DSKO$ D,18,S:S=S+1:GOTO 200
250 N=(S-1)*16+I:DY$=DSKI$(D,18,13)
260 LSET K$="SECRET":LSET N$=CHR$(N)
270 GOTO 460
280 '      *****  Recover  *****
```

```
290 PRINT "Do you recover this secret file ? (y/n) ";:KY$=INPUT$(1)
300 PRINT:IF KY$<>"y" THEN END
310 N=ASC(N$):DIM F1$(N-1)
320 FOR I=0 TO N-1
330 IF I MOD 16=0 THEN J=0:DY$=DSKI$(D,18,12-I¥16)
340 F1$(I)=F$(I MOD 16):NEXT
350 S=1
360 DY$=DSKI$(D,18,S):IF ASC(F$(15))<>255 THEN S=S+1:GOTO 360
370 FOR I=0 TO 15:IF ASC(F$(I))<>255 THEN NEXT:I=I-1
380 FOR J=0 TO N-1:LSET F$(I)=F1$(J)
390 I=I+1:IF I=16 THEN I=0:DSKO$ D,18,S:S=S+1:DY$=DSKI$(D,18,S)
400 NEXT J:DSKO$ D,18,S
410 FOR S=12-(N-1)¥16 TO 12:DY$=DSKI$(D,18,S)
420 FOR I=0 TO 15:LSET F$(I)=STRING$(16,CHR$(255))
430 NEXT I:DSKO$ D,18,S:NEXT S
440 DY$=DSKI$(D,18,13)
450 LSET K$=STRING$(6,CHR$(0)):LSET N$=CHR$(0)
460 DSKO$ D,18,13:PRINT:PRINT "files";D:MOUNT D:FILES D:END
```

実行前のディレクトリの内容(ダンプリストプログラムによる)

```
Drive 2   Track 18   Sector  1   Cluster 24
52 45 46 4F 52 4D 20 20 20 80 23 FF FF FF FF FF  7C9  REFORM    _#・・・・・
56 44 55 4D 50 20 20 20 20 C0 26 FF FF FF FF FF  7ED  VDUMP     ﾀ&・・・・・
53 45 41 52 43 48 42 41 53 80 20 FF FF FF FF FF  827  SEARCHBAS_ ・・・・・
44 49 52 64 6D 70 20 20 20 80 1F FF FF FF FF FF  81A  DIRdmp    _・・・・・・
66 6F 72 6D 61 74 20 20 20 90 29 FF FF FF FF FF  89D  format    ⊥)・・・・・
62 61 63 6B 75 70 20 20 20 90 1E FF FF FF FF FF  87F  backup    ⊥・・・・・・
46 73 68 65 65 74 20 20 20 80 2A FF FF FF FF FF  864  Fsheet    _*・・・・・
4D 45 4D 64 6D 70 20 20 20 80 1D FF FF FF FF FF  818  MEMdmp    _・・・・・・
44 53 4B 64 6D 70 20 20 20 80 2B FF FF FF FF FF  829  DSKdmp    _+・・・・・
```

実行すると、始めにドライブNo.を聞いてきます。

```
run
Drive No. ? 2
Do you make Secret File ? (y/n)

files 2

OK
```

第13セクタのIDの後のキーワードをチェックし、SECRET化されたものであるかどうかを判断し、SECRET FILEにするかどうかを聞いてきます。"y"を入力すると、SECRET化を行なって最後にFILESを実行して、終わります。

実行後のディレクトリを見てみると、

```
Drive 2   Track 18   Sector  1   Cluster 24
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
```

確かにすべてFFになっています。

また、第13セクタを見ると、

```
Drive 2   Track 18   Sector 13   Cluster 25
00 53 45 43 52 45 54 13 FF FF FF FF FF FF FF FF  9D1  .SECRET.........
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
```

第2バイト目からSECRET化されたディスクであることを示すキーワード〝SECRET〞とSECRET化されたファイル数13H(10進数19)が入っていることがわかります。(回復時には、キーワードは消されます。)

この状態で新たなプログラムをSAVEすると、

```
save "2:Test
Ok

files 2
Test   .     1
Ok
```

そのプログラムのみが表示されます。

もう一度ディレクトリを見てみると、

```
Drive 2   Track 18   Sector  1   Cluster 24
54 65 73 74 20 20 20 20 20 80 30 FF FF FF FF FF  7EB  Test      _0.....
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  FF0  ................
```

確かにディレクトリへ書込まれています。

ここで再度このシークレットファイルプログラムを実行すると、

```
run
Drive No. ? 2
Do you recover this secret file ? (y/n)

files 2
Test  .     1    REFORM.    3    VDUMP  .    3    SEARCH.BAS 1    DIRdmp.
format.     1    backup.    1    Fsheet.     1    MEMdmp.    1    DSKdmp.
Error .     1    PTRsub     1    V-list dat 1     FATdsp.    1    STMcpy.
D-edit.     1
OK
```

今度は、RECOVERするかどうかを聞いてきます。"y"を入力すると、直ちに回復作業を行ない、前にSAVEしたプログラムも含めて、FILESにより画面表示して終了します。

この時のディレクトリを見てみると、

```
Drive 2   Track 18   Sector  1   Cluster 24
54 65 73 74 20 20 20 20 20 80 30 FF FF FF FF FF  7EB  Test      _0·····
52 45 46 4F 52 4D 20 20 20 80 23 FF FF FF FF FF  7C9  REFORM    _#·····
56 44 55 4D 50 20 20 20 20 C0 26 FF FF FF FF FF  7ED  VDUMP     タ&·····
53 45 41 52 43 48 42 41 53 80 20 FF FF FF FF FF  827  SEARCHBAS_ ·····
```

前にSAVEした"Test"プログラムに続いて、SECRET化していたファイルが書込まれていることがわかります。

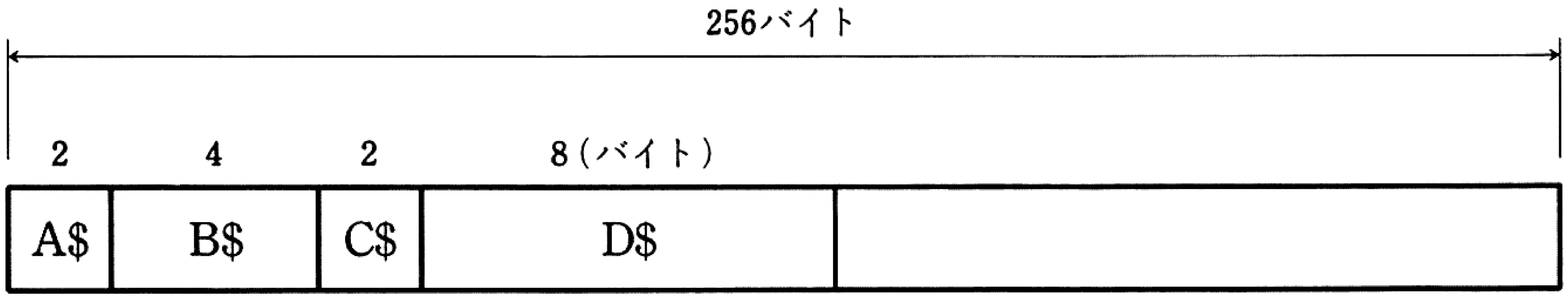
この他の方法もいろいろ考えてみてください。

6-14.FIELD文とファイルデザインシート

Disk BASICにおいて、最高16個(#0～#15)のバッファを使用できますが、1つのバッファは、256バイトの容量があり、データファイル(ランダムアクセスファイル)の1レコード分(1セクタ)にあたります。

この1つのバッファ(1レコード…256バイト)をどのように変数に割り当てるかを指定するのがFIELD文です。

〔図表6-14〕



図表6-14のようにバッファを割当てるときは、

```
100 FIELD#3,2 AS A$,4 AS B$,2 AS C$,8 AS D$
```

注）この場合必ずスペースを設けること。くっつけると、"ASC" という予約語となりSyntax Errorになる。

となります。

FIELD文は、＃0のバッファに対しても宣言可能です。但し、＃0でファイルをOpenはできません。

```
100 FIELD#0,9 AS FN$,1 AS AT$,1 AS CL$
```

FIELD文は、同一バッファに対して何度も宣言できます。

```
100 FIELD #1,10 AS A$,20 AS B$,30 AS C$
110 FIELD #1,15 AS D$,50 AS E$,25 AS F$
```

したがって、1つのレコードを例えば4つに分けてつかう場合、

```
100 FOR I=0 TO 3
110 FIELD #2,I*64 AS DY$,10 AS A$(I),20 AS B$(I),30 AS C$(I)
120 NEXT I
```

ダミーの変数

また、2次元配列変数を利用して、

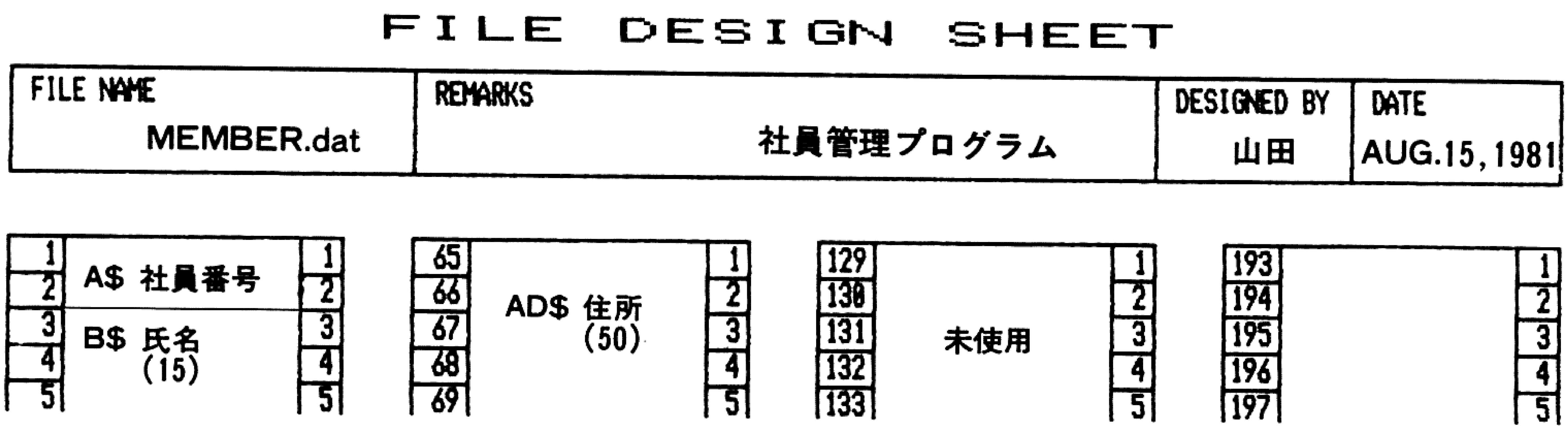
```
100 FOR I=0 TO 3:FOR J=0 TO 7
110 FIELD #1,I*64+J*8 AS DY$,8 AS F$(I,J)
120 NEXT J,I
```

などと、FOR-NEXT文で宣言することもできます。

こうして、FIELD文で定義した変数と領域、およびその内容を、次に示すファイルデザインシートに書込んでおくと、後日プログラムを変更する場合や、同じデータファイルを、他の人が使う場合などに便利です。

〔図表6-15〕 FILE DESIGN SHEET

記入例



このファイルデザインシートをプリンタ（PC-8023）を使って作成するプログラムを示します。

```
100 CLEAR 500:E$=CHR$(27)
110 LPRINT E$"N"E$"L008"E$"A"E$"["CHR$(14)TAB(20)"FILE DESIGN SHEET"CHR$(15)
120 LPRINT E$"Q"E$"T16";
130 LPRINT "┌"STRING$(28,"─")"┬"STRING$(52,"─");
140 LPRINT "┬──────────────┬──────────────┐"
150 LPRINT "| FILE NAME"TAB(29)"| REMARKS";
160 LPRINT TAB(82)"| DESIGNED BY | DATE          |"
170 FOR I=1 TO 2:LPRIN T"|"TAB(29)"|";
180 LPRINT TAB(82)"|"TAB(96)"|"TAB(111)"|":NEXT
190 LPRINT "└"STRING$(28,"─")"┴"STRING$(52,"─");
200 LPRINT "┴──────────────┴──────────────┘":LPRINT
210 LPRINT E$"Q";
220 FOR J=0 TO 3:LPRINT E$"S0200"STRING$(4,CHR$(0))CHR$(3);
230 LPRINT STRING$(31,CHR$(1))CHR$(3)STRING$(135,CHR$(1))CHR$(3);
240 LPRINT STRING$(23,CHR$(1))CHR$(3)STRING$(3,CHR$(0))"     ";:NEXT J
250 LPRINT E$"T04"
260 FOR I=1 TO 64:FOR J=0 TO 3
270 LPRINT USING"|###|                    |##|    ";I+J*64,I;
280 NEXT J:LPRINT E$"T16"
290 IF I=64 THEN 350
300 FOR J=0 TO 3:LPRINT E$"S0200"STRING$(4,CHR$(0))CHR$(7);
310 LPRINT STRING$(31,CHR$(2))CHR$(7)STRING$(135,CHR$(0))CHR$(7);
320 LPRINT STRING$(23,CHR$(2))CHR$(7)STRING$(3,CHR$(0))"     ";:NEXT J
330 LPRINT E$"T06":NEXT I
350 FOR J=0 TO 3:LPRINT E$"S0200"STRING$(4,CHR$(0))CHR$(3);
360 LPRINT STRING$(31,CHR$(2))CHR$(3)STRING$(135,CHR$(2))CHR$(3);
370 LPRINT STRING$(23,CHR$(2))CHR$(3)STRING$(3,CHR$(0))"     ";:NEXT J
```

FILE DESIGN SHEET

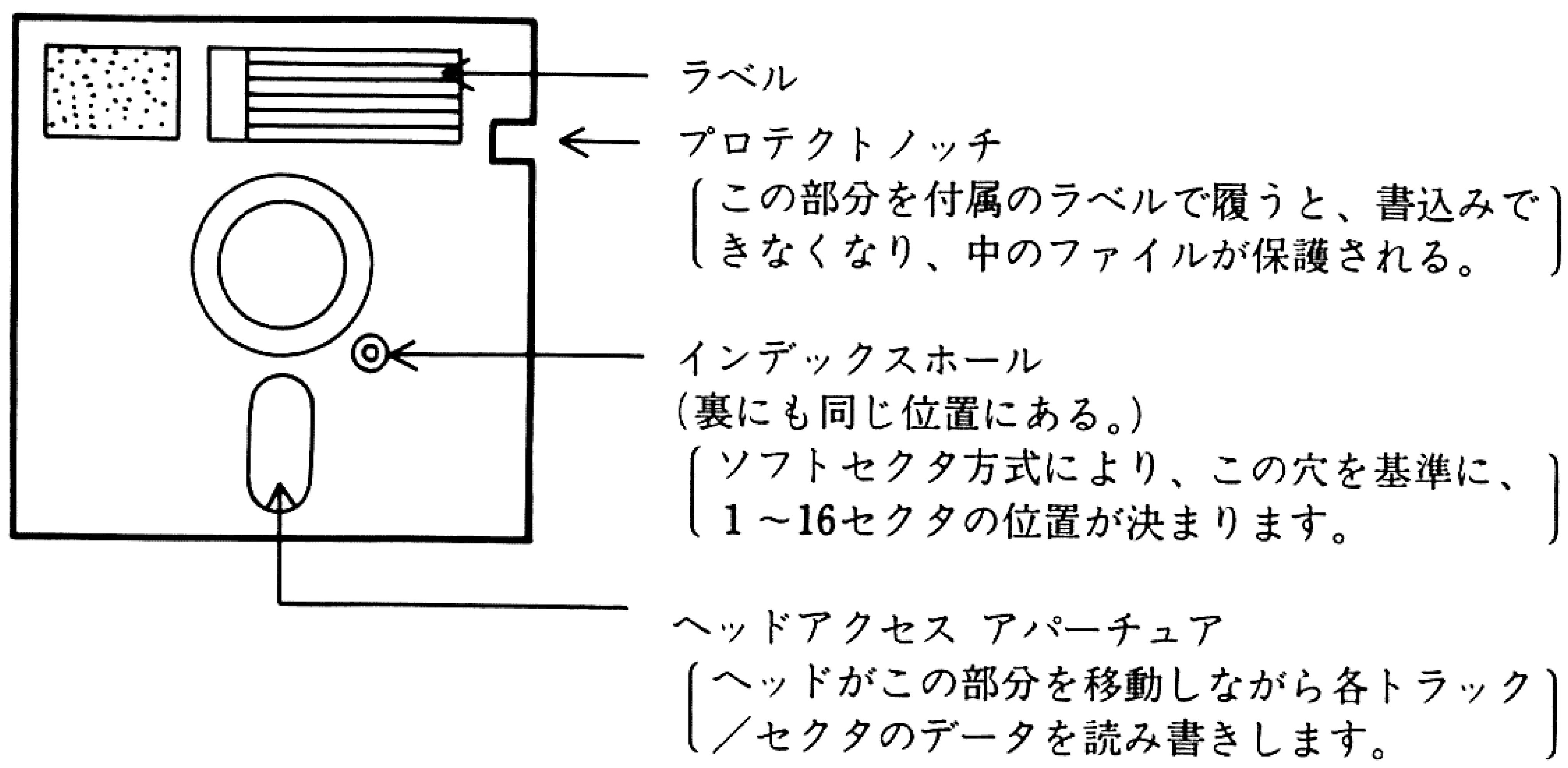
FILE NAME	REMARKS	DESIGNED BY	DATE

1		1	65		1	129		1	193		1
2		2	66		2	130		2	194		2
3		3	67		3	131		3	195		3
4		4	68		4	132		4	196		4
5		5	69		5	133		5	197		5
6		6	70		6	134		6	198		6
7		7	71		7	135		7	199		7
8		8	72		8	136		8	200		8
9		9	73		9	137		9	201		9
10		10	74		10	138		10	202		10
11		11	75		11	139		11	203		11
12		12	76		12	140		12	204		12
13		13	77		13	141		13	205		13
14		14	78		14	142		14	206		14
15		15	79		15	143		15	207		15
16		16	80		16	144		16	208		16
17		17	81		17	145		17	209		17
18		18	82		18	146		18	210		18
19		19	83		19	147		19	211		19
20		20	84		20	148		20	212		20
21		21	85		21	149		21	213		21
22		22	86		22	150		22	214		22
23		23	87		23	151		23	215		23
24		24	88		24	152		24	216		24
25		25	89		25	153		25	217		25
26		26	90		26	154		26	218		26
27		27	91		27	155		27	219		27
28		28	92		28	156		28	220		28
29		29	93		29	157		29	221		29
30		30	94		30	158		30	222		30
31		31	95		31	159		31	223		31
32		32	96		32	160		32	224		32
33		33	97		33	161		33	225		33
34		34	98		34	162		34	226		34
35		35	99		35	163		35	227		35
36		36	100		36	164		36	228		36
37		37	101		37	165		37	229		37
38		38	102		38	166		38	230		38
39		39	103		39	167		39	231		39
40		40	104		40	168		40	232		40
41		41	105		41	169		41	233		41
42		42	106		42	170		42	234		42
43		43	107		43	171		43	235		43
44		44	108		44	172		44	236		44
45		45	109		45	173		45	237		45
46		46	110		46	174		46	238		46
47		47	111		47	175		47	239		47
48		48	112		48	176		48	240		48
49		49	113		49	177		49	241		49
50		50	114		50	178		50	242		50
51		51	115		51	179		51	243		51
52		52	116		52	180		52	244		52
53		53	117		53	181		53	245		53
54		54	118		54	182		54	246		54
55		55	119		55	183		55	247		55
56		56	120		56	184		56	248		56
57		57	121		57	185		57	249		57
58		58	122		58	186		58	250		58
59		59	123		59	187		59	251		59
60		60	124		60	188		60	252		60
61		61	125		61	189		61	253		61
62		62	126		62	190		62	254		62
63		63	127		63	191		63	255		63
64		64	128		64	192		64	256		64

6-15.ディスクの両面使用法

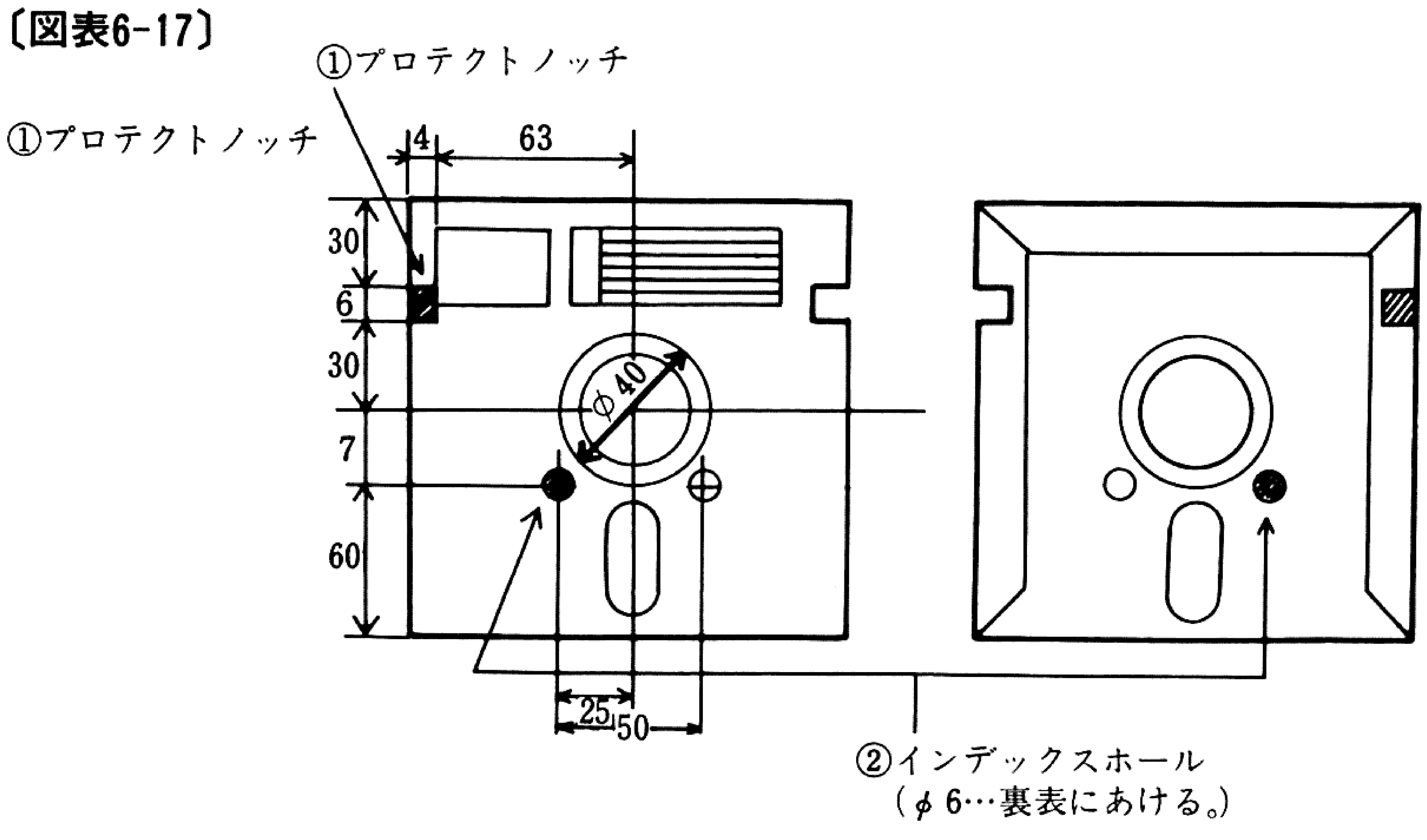
ディスクの外観を下図に示します。

〔図表6-16〕



正方形のカバーの中を回転する薄いフレキシブルディスクには、一般に片面用のディスクであっても両面とも同じように磁性体が塗布されており、次に示す方法で、加工すれば両面使用可能となります。

〔図表6-17〕



(寸法は目安です)

1．プロテクトノッチを加工します。

2．インデックスホールを対称位置に裏表にあける。磁気シートを傷つけぬよう下にプラスチックシートなどを敷いて、カッターナイフなどで慎重にφ6の穴を加工してください。

3．formatプログラムでformatすれば完了。

注）裏面は、その性能を保証されているわけではありませんので、大事なファイルは、表面に、SAVEされた方が無難です。

6-16.ファイルネームの並べ替え

ディスケット内のファイル名の数が多くなると、目的のファイルを探すのに手間どりますが、次のプログラムで、ファイル名をソート（並び替え）しておけば、すぐ見つけ出せます。

```
100 CLEAR2000:DIMFD$(15),F$(70)
110 FORI=0TO15:FIELD#0,I*16ASDY$,16ASFD$(I):NEXT
120 INPUT"Drive No. ";D:N=0:S=1
130 DY$=DSKI$(D,18,S):FOR I=0 TO 15
140 IF ASC(FD$(I))=0 THEN 160
150 F$(N)=FD$(I):IF ASC(F$(N))=255 THEN 180 ELSE N=N+1
160 NEXT:S=S+1:GOTO 130
170 '■■■■  Sort  ■■■■
180 FORI=0TON-2:FORJ=I+1TON-1
190 IF F$(I) > F$(J) THEN SWAPF$(I),F$(J)
200 NEXTJ,I
210 '■■■■  Write  ■■■■
220 FORI=0TON:IF IMOD16=0 THEN DY$=DSKI$(D,18,I¥16+1)
230 LSETFD$(IMOD16)=F$(I)
240 IF IMOD16=15 OR I=N THEN DSKO$ D,18,I¥16+1
250 NEXT:PRINT"Complete"
```

実行前

X-test dat 2	Font dat 3	Laser . 6	Dot dat 2	Inform dat 2
Title dat 2	Pana-C dat 2	P-plan. 1	Bank . 2	E-fare. 2
Cost . 3	Parts . 4	ｿﾝｴｷ . 3	USRLST. 2	Slide . 2
Telex2 dat 1	Telex1 dat 1	Telex . 2	EP-rom. 2	Grid dat 2
5500 dat 2				

実行後

5500 dat 2	Bank . 2	Cost . 3	Dot dat 2	E-fare. 2
EP-rom. 2	Font dat 3	Grid dat 2	Inform dat 2	Laser . 6
P-plan. 1	Pana-C dat 2	Parts . 4	Slide . 2	Telex . 2
Telex1 dat 1	Telex2 dat 1	Title dat 2	USRLST. 2	X-test dat 2
ｿﾝｴｷ . 3				

第7章　USR関数と機械語

7－1　PC-モニタの使い方

7-1-1　モニタの起動法およびBASICへの戻し方

7-1-2　各コマンドの説明

(1) Dコマンド

(2) Sコマンド

(3) Gコマンド

(4) Wコマンド

(5) L, LVコマンド

(6) TMコマンド

(7) 画面クリア

7-1-3　パラメータ

7-1-4　monコマンドのもう1つの使い方

7-1-5　PC-モニタの使い方(図表)

7－2　DEFUSRのしくみ

7－3　機械語サブルーチンの呼び方

7-3-1　BASICからコールする場合

7-3-2　USR関数と引数

(1) 数値型引数

(2) 文字型引数

7－4　BASICを機械語で

7-4-1　ファンクションキー・イニシャライズ

7-4-2　ファンクションキー退避・復活

7-4-3　シークレット・メッセージ

7-4-4　スクリーンコピー

7-4-5　アンリスト

7-4-6　ディスク・オートスタート

第7章　USR関数と機械語

7-1.PCモニタの使い方

機械語を使ううえで必要なものとして、モニタプログラムがあります。これは、直接メモリ内容を操作したり、外部入出力装置とのデータのやりとりを行なうものです。ここでは、PC-8001のモニタの使い方についてマニュアルに書かれていないことにも触れながら解説していきましょう。

PC-8001のモニタプログラムは、N-BASIC ROMの最後(5C2CH～5FFFH)に常駐しています。0.9Kバイトと非常に小さなもので、スクリーンエディットができないなど、不便な点もありますが、使い方に慣れてしまえば、そんなことはありません。

7-1-1 モニタの起動法およびBASICへの戻し方

通常はBASICのコマンド待ちの状態でmon [RET] と入力することにより起動しますが、プログラム中からの起動も可能です。ここで、monの後には何が続いてもよく、例えばmonitorでもかまいません。(ただしこの場合はBASICに戻ったとき、Syntax error が表示されます)

モニタのコマンド待ちの状態で[CTRL] Bを入力するとBASICへ戻りOKが表示されます。ただし、BASICプログラム中からモニタを起動した場合は、もとのプログラムに戻ります。次のプログラムで試してみて下さい。

```
100 ' *** Monitor Test ***
110 PRINT "Now at N-BASIC"
120 PRINT "mon (y/n)?";
130 IF INPUT$(1)<>"y" THEN END
140 PRINT
150 PRINT "Now at Monitor"
160 MON
170 PRINT
180 GOTO 110
```

7-1-2 各コマンドの説明

(1) Dコマンド

メモリの内容を表示するコマンドです。パラメータは、スタートアドレスとエンドアドレスを入力します。エンドアドレスを省略すると、16バイトだけデータが表示されてコマンド待ちに戻ります。

データを表示している途中でやめる場合は、STOPキー、一時止めたい場合は、ESCキーを使うのはBASICの場合と同じですが、ESCキーのリピートはありませんのでいくらか使いやすくなっています。

(2) Sコマンド

メモリの内容を変更するコマンドです。パラメータは、変更したいアドレスを入力します。コマンド入力後、そのアドレスの内容が表示されますので、変更する場合は2ケタの16進数、変更する必要のない場合はスペースを入力していきます。

また、RETキーを押したときや、エラーとなったときは、そのときのアドレスが、カウンタ(FF30H番地、FF31H番地)にセットされますので、続けて行なうときは、S[RET]だけですみます。なおこのカウンタは、他のコマンドの実行や、BASICへ1度戻った後でもそのまま変わりません。

(3) Gコマンド

指定したアドレスにジャンプするコマンドです。パラメータは、ジャンプ先のアドレスを入力しますが、省略すると、0番地になってしまいますので注意が必要です。参考までに、ジャンプするときの各レジスタの値は次のようになっています。

AF＝0A4AH
BC＝5C5EH
DE＝5C82H
HL＝ジャンプするアドレス
SP＝スタックエリアのとり方によって変わる

ジャンプしたプログラムからモニタまたは、BASICに戻したいときは、プログラムの最後を次のようにしておくとよいでしょう。

- モニタへ戻すとき

```
JP   5C66H
```

- BASICへ戻すとき

```
LD   HL,(0FF34H)
RET
```

(4) Wコマンド

カセットテープにメモリの内容を書き込むコマンドです。スタートアドレスとエンドアドレスの2つのパラメータが必要なわけですが、このアドレスはどこでもかまいません。例えばVRAMをセーブ、ロードしてみるというのもおもしろいものです。

このコマンド実行中はSTOPキーがききませんので＊W 0, FFFFのようなことをするとしばらく待たされることになります。

(5) L, LVコマンド

カセットテープからのロードおよびロードベリファイコマンドです。パラメータは

ありません。テープリードエラーおよびSTOPキーで中止した場合は?を表示してコマンド待ちに戻ります。また、LVコマンドでデータに誤りがあった場合も同様です。

(6) TMコマンド

メモリをテストするためのコマンドですがめったに使うものではありません。最近はメモリの性能も向上してきて、よほどひどいものを使わない限りメモリのテストは必要ないはずです。くわしくはユーザーズマニュアルの方を参照して下さい。

(7) 画面クリア

[CTRL] LまたはCLRキーを使うことで画面をクリアします。これでカーソルは左上に移りコマンド待ちになりますが、この時点で[CTRL] Bを入力してもBASICへは戻れません。モニタの唯一のバグでしょうが、ある程度、機械語がわかっている人は、モニタのプログラムをながめてみて、なぜこうなるか考えてみてはいかがですか、(答は簡単なものです)。

7-1-3 パラメータ

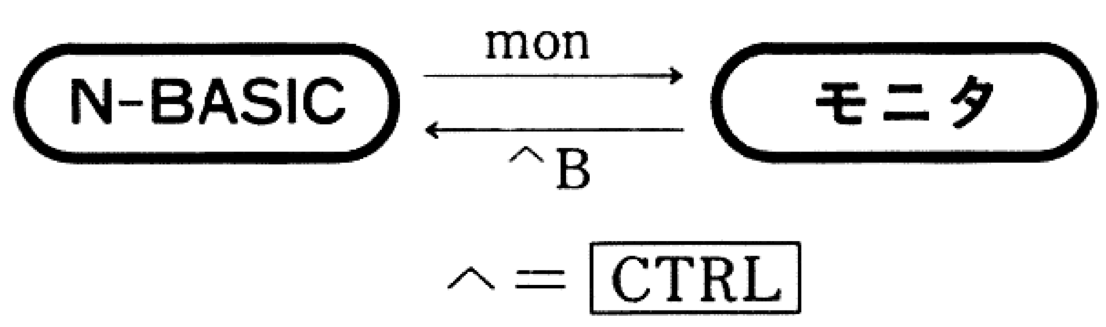
モニタのコマンドのパラメータはすべてアドレスですが、このアドレスの入力は何ケタの数字でもかまいません。4ケタ以上入力した場合は、最後の4ケタが有効になりますので入力誤りをしたときは、続けて入力してやればよいわけです。

7-1-4 monコマンドのもう1つの使い方

monコマンドを実行したとき、7FFFH番地のデータが55Hであれば、7FFCH 番地にジャンプするようになっています。拡張用ROMを使う場合に何かと便利なものです。

7-1-5 PCモニタの使い方(図表)

※モニタの起動と復帰

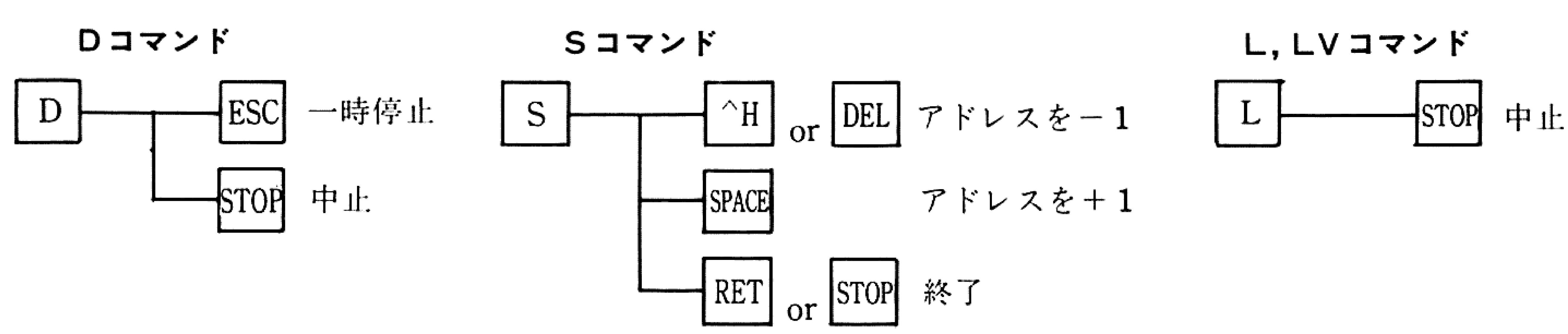


※モニタのコマンド('※'の表示中有効)

コマンド	内　　容	パラメータ
D(Display)	メモリ内容の表示	ADDRESS*, ADDRESS*
S(Set)	メモリへの書込み	ADDRESS*
G(Go)	ジャンプ	ADDRESS*
W(Write)	CMTへのセーブ	ADDRESS, ADDRESS
L(Load)	CMTからロード	なし
LV(Load Verify)	ロードベリファイ	なし
TM(Test Memory)	メモリのテスト	なし
^B	BASICへ戻る	なし
^L or CLR	画面クリア	なし
^M or RET	コマンド待ちに戻る	なし
^J	〃	なし
SPACE	〃	なし
ESC	〃	なし

※　上記以外の文字を入力した場合、またはパラメータに誤りがあると？を表示してコマンド待ちに戻る。
※　パラメータの※付は省略可。

※各コマンド実行中に使えるキー



W, TMコマンド、実行中は何も受け付けない。

＊パラメータの入力

〔例〕Ｄコマンドの場合

＊ＤＣ７Ｃ８００，Ｄ０３０ＣＦＦ０

（有効）（有効）

＝＊ＤＣ８００，ＣＦＦ０

7-2.DEFUSRのしくみ

USR関数の開始番地を指定するのにDEF USR＝&H××××というのも用います。

これは、関数の定義に使うDEFFNとよく似てはいますが、実はたいへんなちがいがあるのです。

まず、DEFFNについてみてみましょう。

DEFFNというのは、ユーザーが関数を定義するためのものですが、次に示すようにダイレクトモードで使うとエラーとなってしまいます。(下線がついているところは、キーボードより入力したもの。以下の例でも同様)

```
def fna(x)=x*x
Illegal direct
Ok
```

また、プログラム中で定義しておいても、プログラムの訂正や抹消を行なうと、他の変数の値と同じように、クリアされてしまいます。

次の例で確かめてみて下さい。

```
10 def fna(x)=sin(x)
20 input x
30 print fna(x)
40 end
run
? 75.5
 .1016
Ok
? fna(32.5)
 .88387
Ok
new
Ok
? fna(32.5)
Undefined user function
Ok
```

ところが、DEFUSRについては、ちゃんとダイレクトモードでも指定可能で、また、プログラム中で一度指定すると、その後プログラムを訂正したりしても、その役目を果します。DEFFNの例と比較してみて下さい。なお、32FCHからはROM内のSIN関数ルーチンです。

```
10 def usr=&H32fc
20 input x
30 print usr(x)
40 end
run
? 75.5
 .1016
Ok
? usr(32.5)
 .88387
Ok
new
Ok
? usr(32.5)
 .88387
Ok
```

つまり、USR関数というのは、DEFUSRで一度指定しておけば、リセットまたは電源を切らない限りその機能が失なわれないわけです。これは、使い方によっては、非常に意味のあるもので、複数のプログラム中で同じ機械語サブルーチンを使うような場合は、最初のプログラムAで、DEFUSRを実行しておけば、次に実行するプログラムBやCにおいては、その必要がなくなるわけです。

それでは、どういうしくみでこうなっているか種明かしをしましょう。

まず、PC本体をリセットして下さい。

それからモニタモードで、＊DEA3B,EA4Eを実行してみて下さい。次の様になっているはずです。

```
NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Microsoft

Ok
mon
*DEA3B,EA4E
EA3B A5 44 A5 44 A5
EA40 44 A5 44 A5 44 A5 44 A5
EA48 44 A5 44 A5 44 A5 44
*
```

次に、BASICに戻ってから、DEFUSRを実行してみて下さい。ダイレクトでも、プログラムでもかまいませんが、ここではダイレクトで行ってみます。

```
defusr0=&H1122
Ok
defusr1=&H3344
Ok
defusr8=&H5566
Ok
defusr9=&H7788
Ok
mon
*DEA3B,EA4E
EA3B 22 11 44 33 A5
EA40 44 A5 44 A5 44 A5 44 A5
EA48 44 A5 44 66 55 88 77
*
Ok
```

```
      auto    go to    list    run
```

先ほどの例と比べてみてどこが変化したところはわかりますね。つまり、DEFUSRは、0～9について、EA3B番地から2バイトずつ飛び先アドレスが書き込まれるわけです。このエリアは、リセットしたときにイニシャライズされるだけですから、USRの指定がずっと残るというわけです。

ここで、頭のいい読者の方ならわざわざDEFSURとしなくても直接このワークエリアに値を書き込めばDEFUSRを実行したのと同じになるのではないかなと思われるでしょう。

実はその通りで次の例で試してみればわかります。

```
NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Microsoft

Ok
mon
*SEA3B
EA3B A5-FC
EA3C 44-32 A5-
*
Ok
print usr(32.5)
 .88387
Ok
```

これは、DEFUSR＝&H32FCを実行したのと同じものです。

ところで、USRを定義する以前にワークエリア上にある44A5というのは何でしょうか。

実は、これは、USRが未定義の場合Illegal function callの番地へ飛ばすためのもので、別に深い意味はありません。

これに関連して、ユーザー関数のクリアのしかたもわかりますね。つまり、DEFUSR＝&H44A5としてやればよいわけです。

7-3.機械語サブルーチンの呼び方

7-3-1 BASICからコールする場合

BASICから機械語ルーチンをコールしたい場合、N-BASICでは、USR関数を用います。USR関数は、DEFステートメントによって、スタートアドレスが設定されます。もしDEFステートメントで、定義せずにUSR関数を用いるとIllegal function callエラーとなりますが、これはイニシャライズのときに、USR関数のスタートアドレスが44A5Hとなっているためです。

USR関数はUSRの後に0から9までの数値をつけることによって最高10種類まで定義することができます。しかし、USR関数を使う前に再定義をしてやれば、別に数字や種類の限度にこだわる必要はありません。ここで、USR関数というものを実際のプログラムを通してみることにしましょう。

まず、機械語ルーチンが、関数として働く例として次のプログラムをあげておきます。

```
list
1000 REM *** usr function ***
1010 DEFUSR=&H32FC
1020 TH=45:RD=3.14*TH/180
1030 PRINT "sin(";TH;"°) =";SIN(RD)
1040 PRINT "usr(";TH;"°) =";USR(RD)
Ok
run
sin( 45 °) = .706825
usr( 45 °) = .706825
Ok
```

この例では、機械語ルーチンとしてROM内のSIN関数のところを使ったわけですが、これで、なぜ機械語ルーチンをコールする場合、USRを使いUSR関数と呼ばれる意味がわかるかと思います。

次に、機械語ルーチンを関数として使うのではなく、単なるサブルーチンとして使う例を示します。

なお、この章で掲載するサンプル・プログラムは、次の3つの部分に分かれています。

① 機械語ルーチン

② BASICプログラム

③ 機械語とアセンブラ・リスト

アセンブラは、ザイログ社Z-80文法に準拠しています。なお、アセンブラの各命令の意味については、ここでは説明を省いており、Z-80アセンブラに関する他の参考書に譲ることにします。

```
mon
*DE800,E816
E800 5E 23 56 42 4B 3E 20 D3
E808 40 0D 20 FD 4B AF D3 40
E810 0D 20 FD 1D 10 EE C9
*
```

（プログラム1.1）

```
1000 REM ***** Sound *****
1010 CLEAR 300,&HE7FF
1020 DEFUSR=&HE800
1030 INPUT HE$
1040 FOR I=0 TO 4
1050    DUMMY=USR(VAL("&H"+HE$))
1060 NEXT
1070 GOTO 1030
```

（プログラム1.2）

このプログラムは、BEEPの音を使っていろいろな音色を出すものです。モニタから機械語を入力して、BASICのプログラムを実行すると、'？'が画面上に現れますので、4ケタの16進数を入力してみて下さい。どうですか、これは機械語だからこそできる技なのです。

```
************************
*  SOUND   ************
************************
                   ;
                           ORG   0E800H
                   ;
E800 5E            SOUND:  LD    E,(HL)      入力された16進数の下位バイトを
                                             Eレジに。E=PEEK(HL)
E801 23                    INC   HL          HL =HL+1
E802 56                    LD    D,(HL)      上位バイトをDレジに。
                                             D=PEEK(HL)
E803 42                    LD    B,D         B=D（Bレジを外側ループのカ
                                             ウンタとしてセット）
E804 4B            LOOP:   LD    C,E         C=E（Cレジを内側ループのカ
                                             ウンタとしてセット）
E805 3E20                  LD    A,20H
E807 D340                  OUT   (40H),A     BEEP 1
E809 0D            L1:     DEC   C           C=C－1
E80A 20FD                  JR    NZ,L1       IF C<>0 THEN GOTO L1
E80C 4B                    LD    C,E         C=E
E80D AF                    XOR   A           A=0
E80E D340                  OUT   (40H),A     BEEP 0
E810 0D            L2:     DEC   C           C=C－1
E811 20FD                  JR    NZ,L2       IF C<>0 THEN GOTO L2
E813 1D                    DEC   E           E=E－1
E814 10EE                  DJNZ  LOOP        B=B－1：IF B<>0 THEN
                                             GOTO LOOP
E816 C9                    RET               RETURN
                   ;
E817                       END
```

機械語サブルーチンをコールするのになぜ、関数をと思う人もいるかもしれませんが、今の例のように代入文の右辺にダミー変数をもってくることによって立派にサブルーチンコールができるわけです。

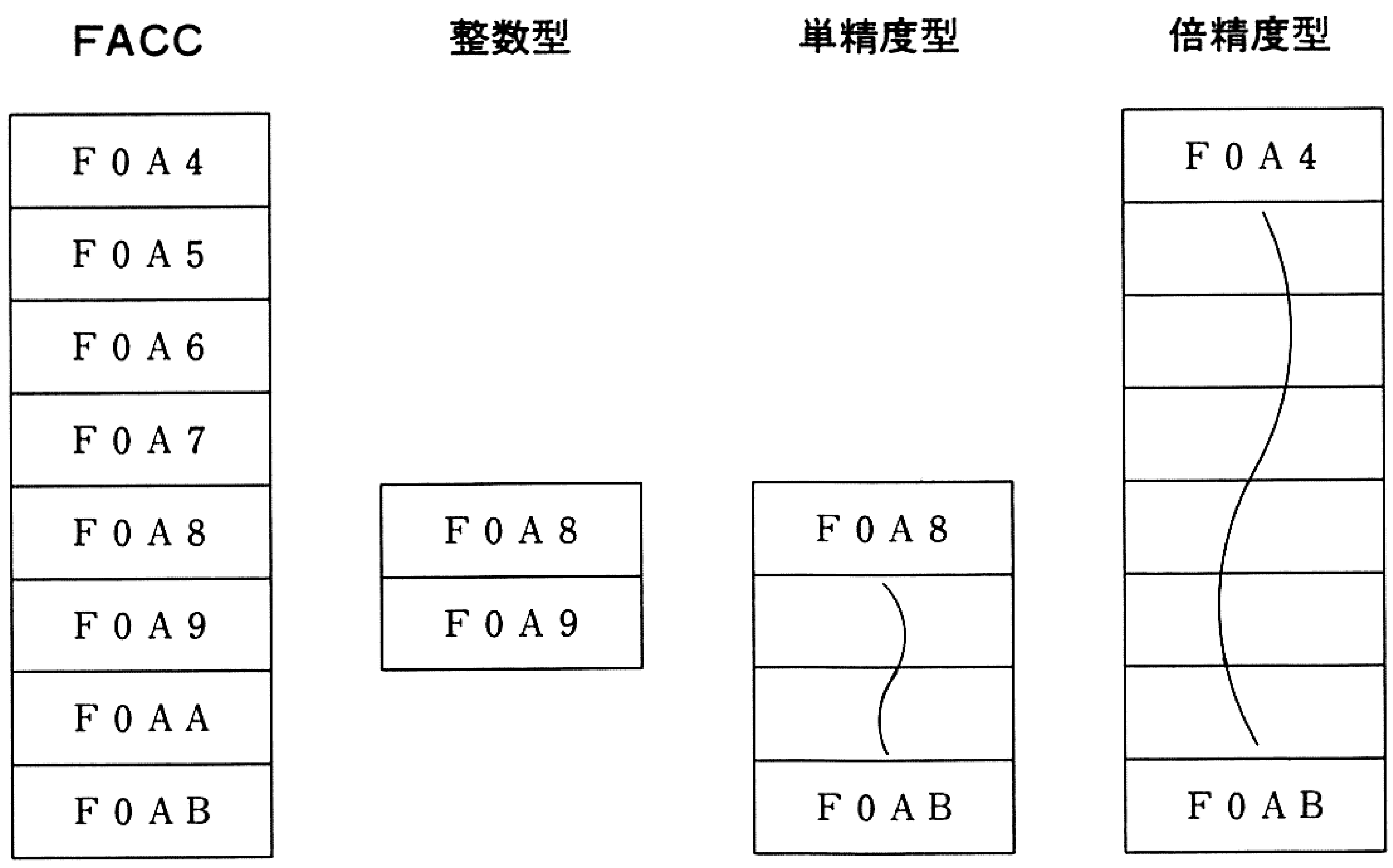
7-3-2 USR関数と引数

USR関数は機械語プログラムを呼ぶだけでなく、値をわたすこともできます。これは、引数と呼ばれ、数値型（整数、単精度実数、倍精度実数）と文字型のものがあります。

(1) 数値型引数

数値型の引数は、浮動小数点アキュームレータ（以下FACCと略します）を通してわたされます。FACCはワークエリア上（F0A4H～F0ABH）にあり、数値の型によって、次のように格納されています。

〔図表7-2〕　引　数



なお、引数は、HLレジスタが指すアドレスから格納され、USR関数が呼び出されると、HLレジスタの値はF0A8Hとなります。

ここでは、例として整数型へ引数をわたす場合をあげておきます。次のプログラムは、エラーコードを引数として、そのエラーメッセージをプリントするプログラムです。

```
mon
*DE800,E818
E800 5E 21 F1 37 7B FE 1E 38
E808 02 1E 15 CD C0 45 1D 23
E810 20 F9 7E B7 C8 DF 23 18
E818 F9
*
```

```
1000 REM ***** ERROR MESSAGE *****
1010 CLEAR 300,&HE7FF
1020 DEFUSR=&HE800
1030 INPUT "ERROR CODE";EC%
1040 DUMMY=USR(EC%):PRINT
1050 GOTO 1030
```

```
*********************
*  ERROR MES  *******
*********************
                   ;
                         ORG   0E800H
                   ;
E800 5E            ERMES: LD    E,(HL)       ; E:=(0F0A8H) Error Code
E801 21F137               LD    HL,37F1H     ; Error mes top アドレス
E804 7B                   LD    A,E            A=E
E805 FE1E                 CP    1EH
E807 3802                 JR    C,SERCH        IF A-1EH < 0 THEN SERCH
E809 1E15                 LD    E,15H          ELSE E=15H (Unprintable error)
E80B CDC045        SERCH: CALL  45C0H
E80E 1D                   DEC   E              エラーコードに対するメッセージの
E80F 23                   INC   HL             格納アドレスをサーチ
E810 20F9                 JR    NZ,SERCH
E812 7E            PNEXT: LD    A,(HL)         HL=メッセージ開始アドレス
E813 B7                   OR    A
E814 C8                   RET   Z            ; End of string (IF A=0 THEN RETURN)
E815 DF                   RST   18H          ; Output one character (1文字出力)
E816 23                   INC   HL             HL=HL+1
E817 18F9                 JR    PNEXT          GOTO PNEXT
                   ;
E819                      END
```

エラーコードは1バイトですみますので、引数を整数型として、その下位バイトをEレジスタにわたすようにしています。2バイトともわたしたいときは、プログラムの先頭で、

```
LD   E, (HL)
INC  HL
LD   D, (HL)
```

としてやれば、DEレジスタに整数の値がはいります。

(2) 文字型引数

文字列が引数となるときは、ワークエリア上のストリングディスクリプタによってひきわたされます。ストリングディスクリプタとは文字列の長さ、格納アドレス(下位・上位)を示し、この先頭アドレスは、DEレジスタにより指されています。(通常、DEレジスタの値は、EF58Hです)。ここで示す例は、文字列を引数としてわたし、その文字列を逆にプリントするプログラムです。

```
mon
*DE800,E811
E800 EB 4E 23 5E 23 56 EB 06
E808 00 09 2B 41 7E DF 2B 10
E810 FB C9
*
```

```
1000 REM ***** INV STRING *****
1010 CLEAR 300,&HE7FF
1020 DEFUSR=&HE800
1030 INPUT "STRING";ST$:IF ST$="" THEN 1030
1040 DUM$=USR(ST$):PRINT
1050 GOTO 1030
```

```
*******************
*  INV STRING *****
*******************
                 ;
                         ORG   0E800H
                 ;
E800 EB          IVSTR:  EX    DE,HL              HL=VARPTR(ST$)
E801 4E                  LD    C,(HL)          ;  C:=Length =PEEK (HL)
E802 23                  INC   HL                 HL=HL+1
E803 5E                  LD    E,(HL)             E =PEEK (HL)
E804 23                  INC   HL                 HL=HL+1
E805 56                  LD    D,(HL)             D=PEEK (HL)
E806 EB                  EX    DE,HL           ;  HL:=Top of string
E807 0600                LD    B,0                B=0
E809 09                  ADD   HL,BC              HL=HL+BC
E80A 2B                  DEC   HL                 HL=HL-1(文字列の最後のアドレスをセット)
E80B 41                  LD    B,C                B=C(文字の長さをカウンタとしてセット)
E80C 7E          LOOP:   LD    A,(HL)
E80D DF                  RST   18H             ;  Print one character
E80E 2B                  DEC   HL                 HL=HL-1
E80F 10FB                DJNZ  LOOP               B=B-1:IF B<>0 THEN LOOP
E811 C9                  RET                      RETURN
                 ;
E812                     END
```

USR関数では、引数を機械語ルーチンにわたすだけでなく、BASICの方へ値をもどすこともできます。最初に示したSIN関数などが良い例です。結果を戻すには、原理的にはFACCとストリングディスクリプタおよび、FACCの型を示すワークエリアに値を代入してやります。引数と結果が同じ型であれば、数値型についてはFACC, 文字型についてはストリングディスクリプタに値を戻すだけですみます。また、型が異なる場合でも、同じ数値型同志ならFACCのタイプを示すワークエリア(EF45H)に値を入れれば可能です。

ここでやっかいなのは、引数と結果が、数値型と文字型の関係のときです。上記の3つのワークエリアを設定するのに加えて、ストリングディスクリプタのスタックを扱うなど手間がかかります。次に示す例ではROM内のルーチンにまかせていますが、これが最も簡単で確実な方法でしょう（52A9Hは、HLレジスタで文字列の先頭を指してコールすると、上の処理をすべてやってくれるルーチンです。文字列→文字列でも使えます)。

ここでは、引数と結果の型が同じものの例として、文字列のうち小文字のアルファベットを大文字にして返すプログラムを、引数と結果の型が異なるものとして、エラーメッセージを文字列として返すプログラムをあげておきます。

```
mon
*DE800,E814
E800 D5 EB 46 23 5E 23 56 EB
E808 D1 78 B7 C8 7E CD C1 5F
E810 77 23 10 F8 C9
*
```

```
1000 REM ***** Conv a-z *****
1010 CLEAR 300,&HE7FF
1020 DEFUSR=&HE800
1030 INPUT "String";ST$
1040 CV$=USR(ST$)
1050 PRINT CV$
1060 GOTO 1030
```

```
*******************
* a-z => A-Z ******
*******************
                   ;
                          ORG   0E800H
                   ;
E800 D5            AZCONV:PUSH  DE          DEの値を退避
E801 EB                   EX    DE,HL       HL=VARPTR (ST$)
E802 46                   LD    B,(HL)     ; B:=Length =PEEK(HL)
E803 23                   INC   HL          HL=HL+1
E804 5E                   LD    E,(HL)      E =PEEK (HL)
E805 23                   INC   HL          HL=HL+1
E806 56                   LD    D,(HL)      D =PEEK (HL)
E807 EB                   EX    DE,HL      ; HL:=Top of string
```

```
E808 D1                POP   DE      DEの値を復旧(文字列のポインタをリセット)
E809 78                LD    A,B  ⎫ IF A=0 THEN RETURN
E80A B7                OR    A    ⎬
E80B C8                RET   Z    ⎭ 文字列がヌルならBASICへ
E80C 7E        LOOP:   LD    A,(HL)
E80D CDC15F            CALL  5FC1H          ; Convert (大文字変換)
E810 77                LD    (HL),A           大文字をもとのアドレスにセット
E811 23                INC   HL               HL=HL+1
E812 10F8              DJNZ  LOOP             B=B−1 : IF B<>0 THEN LOOP
E814 C9                RET                    ELSE RETURN
               ;
E815                   END
```

```
mon
*DE800,E815
E800 5E 21 F1 37 7B FE 1E 38
E808 02 1E 15 CD C0 45 1D 23
E810 20 F9 CD A9 52 C9
*
```

```
1000 REM ***** ERROR MES (2) *****
1010 CLEAR 300,&HE7FF
1020 DEFUSR=&HE800
1030 INPUT "Error code";EC%
1040 MSG$=USR(EC%)
1050 PRINT MSG$
1060 GOTO 1030
```

```
********************
*  ERROR MES (2) ****
********************
               ;
                       ORG   0E800H
               ;
E800 5E        ERMES:  LD    E,(HL)         ; E:=(0F0A8H) Error Code
E801 21F137            LD    HL,37F1H       ; Error mes top
E804 7B                LD    A,E
E805 FE1E              CP    1EH
E807 3802              JR    C,SERCH
E809 1E15              LD    E,15H
E80B CDC045    SERCH:  CALL  45C0H
E80E 1D                DEC   E
E80F 23                INC   HL
E810 20F9              JR    NZ,SERCH
               ;
E812 CDA952            CALL  52A9H          ; 引数と結果の処理ルーチン
E815 C9                RET
               ;
E816                   END
```

7-4.BASICを機械語で

この稿では、本書で取り扱ったBASICプログラムを機械語で書いた場合のプログラム例を紹介するとともに、有用な機械語ルーチンおよび未使用コマンドの使い方を説明します。

7-4-1 ファンクションキー・イニシャライズ

第3章キー入力では、ファンクションキーのイニシャライズ(初期設定)をする際、次のステートメントを実行しました。

```
10 FOR I=0 TO 159:POKE&HEA7C+I,PEEK(&H3A81+I):NEXT I
```

これは、3A81番地から格納されている160バイト分の内容を、そのまま順次EA7C番地から書き込んでいます。同様のことを機械語で行なうには、ブロック転送（ある番地のメモリの内容を別の番地にそのまま移動させる)の命令を実行します。

ブロック転送を行なうためには、Z-80の3つのレジスタペア(HL,DE,BC)を次のように初期設定しておく必要があります。

HL＝転送する内容の開始番地
DE＝転送先の開始番地
BC＝転送するバイト数

そして、ブロック転送命令LDIRを使います。これを実行すると、HLが指す番地の内容をDEが指す番地に転送されます。そして、HL＝HL＋1、DE＝DE＋1、BC＝BC－1が実行され、BCの値が0になるまで続けられます。次に、アセンブラ・リストを紹介します。

```
LD   HL, 3A81H
LD   DE, EA7CH
LD   BC, A0H  (10進で160バイト)
LDIR
```

この3つのレジスタペアのロード命令は覚えやすく、順に、21、11、01となります。また、LDIRは、EDB0で、ブロック転送後、BASICのコマンド待ち(3C82H）にジャンプするようにします。それでは、プログラム例を次に紹介します。モニタ・モードでキーインした後、GFF40 [RETURN] で実行してください。

```
mon
*DFF40,FF4D
FF40 [21] 81 3A [11] 7C EA [01] A0
FF48 00 ED B0 C3 82 3C
*
Ok
```

```
************************
* FUNCTION KEY INIT ****
************************
                ;
                        ORG   0FF40H
                ;
FF40 21813A             LD    HL,3A81H      ; ADDRESS OF THE FIRST SOURCE BYTE
FF43 117CEA             LD    DE,0EA7CH     ; ADDRESS OF THE FIRST DESTINATION BYTE
FF46 01A000             LD    BC, 0A0H      ; NUMBER OF BYTES TO BE MOVED
FF49 EDB0               LDIR                ; BLOCK TRANSFER
FF4B C3823C             JP    3C82H         ; JUMP TO BASIC COMMAND LEVEL
                ;
FF4E                    END
```

7-4-2 ファンクションキー退避・復活

次の例では、定義されたキーの内容をあるメモリ領域に退避させ、保存しておき、必要な時に復活させるというものです。

退避はA＝USR(1)、復活はA＝USR(0)で行ないますが、ポイントは、これらの引数をAレジスタに入れ、そのORをとり、0かそうでないかを判定し、ブロック転送の転送先を変更していることです。

ファンクションキー退避、復活プログラム
（機械語　バージョン）

ソースプログラム

```
********************
* FUNTION KEY ******
********************
                ;
FF40            START:  EQU   0FF40H
                ;
EA7C            FUNCKY:EQU    0EA7CH
00A0            LENGTH:EQU    160
                ;
                        ORG   START
                ;
FF40 7E         USR:    LD    A,(HL)        A＝引数（0か1）
FF41 217CEA             LD    HL,FUNCKY     HL＝EA7CH
FF44 1151FF             LD    DE,FUNCBF     DE＝FF51H
FF47 B7                 OR    A
FF48 2001               JR    NZ,L1         IF A<>0 THEN L1
FF4A EB                 EX    DE,HL         ELSE SWAP DE, HL
FF4B 01A000     L1:     LD    BC,LENGTH     BC＝160
FF4E EDB0               LDIR                ブロック転送
FF50 C9                 RET                 RETURN
                ;
FF51            FUNCBF:DS     LENGTH        FF51Hから160バイト確保
                ;
FFF1                    END
```

```
100 '        ※※※ Function Key ※※※
110 '
120 '      Machine Language Version
130 '        using  FF40H - FFF0H
140 '
150 '     DEF USR=&HFF40
160 '     A=USR(1)   function key ﾀｲﾋ          } 使い方
170 '     A=USR(0)   function key ﾌｯｶﾂ         }
180 '
190 FOR AD=&HFF40 TO &HFF50
200 READ DA$
210   POKE AD,VAL("&H"+DA$)
220 NEXT
240 DATA 7E,21,7C,EA,11,51,FF,B7,20,1,EB,1,A0,0,ED,B0,C9
```

7-4-3 シークレット・メッセージ

機械語で任意の文字列を画面に表示するにはどうしたらよいのでしょう？　答えは簡単です。N-BASICのROM内に文字列出力ルーチンがあり、これを利用すればよいのです。

HLレジスタに文字列が格納されている開始番地を入れ、文字列出力ルーチン52EDHをコールします。ここで1つ注意することは、文字列を格納した際、その終りの印として、00〔CHR$(0)〕を最後に付けておく必要があることです。

試しに、FF50Hから次のアスキーコード（文字列）を入れて、HLレジスタにFF50Hをロードし、52EDHをコールしてください。このプログラムはFF40Hから入れ、GFF40で実行することにします。PC-Tech Know 8000と表示されます。

```
                        ORG   0FF40H
                    ;
FF40  2150FF            LD    HL,0FF50H    HL＝文字列アドレス
FF43  CDED52            CALL  52EDH        文字列出力ルーチン
FF46  C3823C            JP    3C82H        BASICへジャンプ
                    ;
FF49                    END
```

```
mon
*DFF40
FF40 21 50 FF CD ED 52 C3 82 3C 00 00 00 00 00 00 00
*
*
*
mon
*DFF50,FF60
FF50 50 43 2D 54 65 63 68 4B 6E 6F 77 20 38 30 30 30
FF60 00
*
*
*
*
```

さて、次のメッセージは何でしょうか？シークレット・メッセージとしておきますので、読者が出力してみてください。なお、メッセージを1字1字、アスキーコードに変換し、メモリに書き込むのはめんどうですので、次頁のBASICプログラムを用いると便利です。

```
mon
*DFF40
FF40 21 50 FF CD ED 52 C3 82 3C 00 00 00 00 00 00 00
*
*
*DFF50,FFD0
FF50 86 86 86 86 20 54 68 61 6E 6B 20 79 6F 75 20 66
FF60 6F 72 20 72 65 61 64 69 6E 67 20 50 43 2D 54 65
FF70 63 68 4B 6E 6F 77 20 38 30 30 30 20 86 86 86 86
FF80 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
FF90 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
FFA0 2A 2A 2A 2A 20 59 6F 75 20 61 72 65 20 6F 6E 65
FFB0 20 6F 66 20 74 68 65 20 67 72 65 61 74 65 73 74
FFC0 20 50 43 20 75 73 72 73 2E 20 2A 2A 2A 2A 2A 2A
FFD0 00
*
```

シークレット・メッセージ

```
10 REM *** SECRET MESSAGE ***
20 LINEINPUT"MESSAGE ? ";A$:A$=A$+CHR$(0)
30 FOR I=0 TO LEN (A$)-1
40   POKE &HFF50+I,ASC(MID$(A$,I+1,1))
50 NEXT
60 END
```

メッセージ書き込みプログラム

7-4-4 スクリーンコピー

今度は、Disk BASICの未使用コマンドを用いて、スクリーンコピーを行なうプログラムを紹介します。コマンドは、CMDを用いることにします。CMDのジャンプ・アドレスはF0FCHで、そこから3バイトは、C3·75·18となっています。1875Hは、Disk BASIC Featureのエラーを出力してコマンド待ちに戻るルーチンですので、1875Hを、スクリーンコピー・プログラムを格納するアドレスに変更します。ここでは、FF40Hとし、F0FDHに40H、F0FEHにFFHを書き込みます。

```
mon
*SF0FC
F0FC C3-   75-40  18-FF  C3-
```

次に、FF40Hからスクリーンコピーのルーチンを書きます。この時、HLの値を保存しておいてルーチンの最後はRET(C9)とすると、CMDコマンド実行後、BASICへ戻ってきます。

```
mon
*DFF40
FF40 E5 CD 4A 12 E1 C9 00 00 3C 00 00 00 00 00 00 00
*
OK
```

```
**********************
* SCREEN COPY 'CMD' **
**********************
                  ;
                        ORG   0FF40H
                  ;
FF40 E5                 PUSH HL
FF41 CD4A12             CALL  124AH          ; SCREEN COPY ROUTINE
FF44 E1                 POP   HL
FF45 C9                 RET                  ; RETURN TO BASIC
                  ;
FF46                    END
```

7-4-5 アンリスト

次のサンプル・プログラムは、LISTコマンドを実行しても、ただOKを表示して、プログラム・リストがとれませんが、その実行はできるというものです。

BASICプログラムの開始番地は16Kの場合、C021H、32Kの場合、8021Hからで、8023Hから2バイトは、最初の行の行番号が格納されています。この2バイトをFFHとして、初期化すればOKです。ここでは32Kを例にとります。

コマンドは、やはり未使用コマンドを用い、リストを消すにはISET、復活させるにはIRESTと定めます。ISETのジャンプアドレスがF105H、IRESETがF108Hですので、これらのアドレスにそれぞれの処理ルーチンアドレスをここではFF40H、FF5DHと設定します。

```
mon
*SF105
F105 C3-   75-40 18-FF
F108 C3-   75-5D 18-FF C3-
*
```

プログラムはエラー・チェックがなされており、リストが消されているのにISETしたり、リストが復活しているのにIRESETすると、Illegal function callのエラーとなります。

```
mon
*DFF40,FF7F
FF40 21 23 80 11 80 FF 7E FE FF 20 05 1E 05 C3 F9 3B
FF50 12 23 13 7E 12 3E FF 77 2B 77 C3 82 3C 21 23 80
FF60 7E FE FF 28 05 1E 05 C3 F9 3B 11 80 FF 1A 77 23
FF70 13 1A 77 C3 82 3C 00 00 00 00 00 00 00 00 00 00
*
```

```
*********************
* UNLIST 'ISET・IRESET
*********************
                ;
                        ORG     0FF40H
                ;
FF40 212380     ISET:   LD      HL,8023H        ; HL=First Line Number
FF43 1180FF             LD      DE,0FF80H       ; Work Area,Save Line Number
FF46 7E                 LD      A,(HL)            Aに行番号の下位バイトをセット
FF47 FEFF               CP      0FFH            ; IF FFH then Error
FF49 2005               JR      NZ,UNLIST
FF4B 1E05               LD      E,5H            ; Illegal function call
FF4D C3F93B             JP      3BF9H           ; Error message output routine
                ;
FF50 12         UNLIST:LD       (DE),A            FF80Hに行番号の下位バイトをセット
FF51 23                 INC     HL
FF52 13                 INC     DE
FF53 7E                 LD      A,(HL)
FF54 12                 LD      (DE),A            FF81Hに行番号の上位バイトをセット
```

```
                    ;
FF55 3EFF               LD    A,0FFH         ; Initialize line number
FF57 77                 LD    (HL),A           8024HにFFHをセット
FF58 2B                 DEC   HL
FF59 77                 LD    (HL),A           8023HにFFHをセット
FF5A C3823C             JP    3C82H          ; JUMP TO BASIC
                    ;
FF5D 212380         IRESET:LD  HL,8023H
FF60 7E                 LD    A,(HL)
FF61 FEFF               CP    0FFH           ; IF FFH Error
FF63 2805               JR    Z,LIST
FF65 1E05               LD    E,5H           ; Illegal function call
FF67 C3F93B             JP    3BF9H          ; Error message output routine
                    ;
FF6A 1180FF         LIST:   LD    DE,0FF80H      FF80, 81Hに退避していた
FF6D 1A                 LD    A,(DE)           行番号を8023, 24Hにセット
FF6E 77                 LD    (HL),A
FF6F 23                 INC   HL
FF70 13                 INC   DE
FF71 1A                 LD    A,(DE)
FF72 77                 LD    (HL),A
FF73 C3823C             JP    3C82H
                    ;
FF76                    END
```

7-4-6 ディスク・オートスタート

Disk BASICで、プログラムをRUNまたはLOADするには、ディスクシステムの起動(スイッチオンまたはリセット)、システムの読み込み、OPENするファイルの数の指定、MOUNT、目的のファイル名の入力という一連のプロセスが必要ですが、これらの作業をすべてPCにまかせ、プログラムをオートスタートさせることができます。

PCをスイッチオンした時、ディスクが接続されていればN-BASICのIPLプログラムによって、ドライブ1に入っているシステムディスケットのトラック0、セクタ1の内容がPC本体のC000H～C0FFHにLOADされ、C000Hから実行されます。これにより、トラック0、セクタ2～6、トラック1～2の内容がEFFFHまでロードされます。オートスタートにするには、トラック0のセクタ1と2の内容を少し変更して、システムディスケットに書き込んでおけばOKです。

さて、ここで問題になることが2つあります。まず1つは、OPENするファイルの数の指定、そして、もう1つはRUNさせるファイル名の決定です。この例では、ファイル数は3、ファイル名はBOOTとすることにします。

Disk BASICでは、OPENする数は画面にHow many files(0～15)?と表示して、キーインさせる方法をとっています。この部分のプログラムの逆アセンブラ・リストを紹介し説明しましょう。

```
C06E-   21 03 C1        LD    HL,C103H
C071-   CD ED 52        CALL  52EDH      ───→ PRINT" How many files"
C074-   CD 5C 3E        CALL  3E5CH      ───→ INPUT A
                                              （ファイル数の入力）
C077-   38 F5           JR    C,C06EH    ───→ 入力ミスの際C06EHにジャンプ（文字）
C079-   D7              RST   10H
C07A-   B7              OR    A
C07B-   3E 05           LD    A,05H      ───→ RET キーのみを押したときのファイル数（5）
C07D-   28 0E           JR    Z,C08DH    ───→ RET キーのみを押したときC08DHにジャンプ
C07F-   CD D5 4C        CALL  4CD5H
C082-   20 EA           JR    NZ,C06EH  ┐
C084-   7A              LD    A,D       │
C085-   B7              OR    A         │
C086-   20 E6           JR    NZ,C06EH  ├ 入力ミスの際、C06EHにジャンプ（数>15、,など）
C088-   7B              LD    A,E       │
C089-   FE 10           CP    10H       │
C08B-   30 E1           JR    NC,C06EH  ┘
C08D-   32 D4 E9        LD    (E9D4H),A  ───→ ファイルの数を格納
```

C06EH～C073Hで"How many files"を表示して、キー入力待ちにします。オートスタートさせるには、この部分は不要ですのでNOP(00)に変更します。また、C07DH～C08CHまでは入力ミスの際のチェック・ルーチンですので、この部分もNOPに変更します。次に変更後のリストを示します。

```
C06E-   00              NOP
C06F-   00              NOP
C070-   00              NOP
C071-   00              NOP
C072-   00              NOP
C073-   00              NOP
C074-   00              NOP
C075-   00              NOP
C076-   00              NOP
C077-   00              NOP
C078-   00              NOP
C079-   00              NOP
C07A-   00              NOP
C07B-   3E 03           LD    A,03H      ファイル数3と指定
C07D-   00              NOP
C07E-   00              NOP
C07F-   00              NOP
C080-   00              NOP
C081-   00              NOP
C082-   00              NOP
C083-   00              NOP
C084-   00              NOP
C085-   00              NOP
```

```
C086-   00              NOP
C087-   00              NOP
C088-   00              NOP
C089-   00              NOP
C08A-   00              NOP
```

★この部分は、トラック0，セクタ1に入っています。

次に、ファイル名ですが、本書第3章キー入力のファンクションキー・フラグの使い方と同様のことを機械語で行ないます。

mount 1：run〝BOOT〞C/RというコマンドをC110Hから入れておき、EDC0H、EDC1Hにそのアドレスを設定し、EA68Hに1を入れます。このとき、HLレジスタを使用するので、HLの値をPUSHして保存し、処理後、POPで戻して、BASICへジャンプさせます。以下にリストを示します。

```
C100-   E5              PUSH HL
C101-   21 10 C1        LD   HL,C110H
C104-   22 C0 ED        LD   (EDC0H),HL    コマンド格納アドレス
C107-   3E 01           LD   A,01H
C109-   32 68 EA        LD   (EA68H),A
C10C-   E1              POP  HL            ファンクションキー・フラグ
C10D-   C3 C8 17        JP   17C8H         を立てる。
```

```
          m  o  u  n  t  1  :  r  u  n  "  B  O  O  T  "
DC110,C121
C110 6D 6F 75 6E 74 31 3A 72 75 6E 22 42 4F 4F 54 22
C120 0D 00
*    ↑  ↑
*    C/R コマンドの後り
```

★この部分は、トラック0、セクタ2に入っています。

これで、システムのスイッチオンまたはリセットで、BOOTというプログラムが自動的にRUNします。なお、ディスケットの内容変更は、第6章のディスク・エディット・プログラムを使えば、簡単に行なえます。

第8章　ランダム・テクニック

第8章　ランダム・テクニック

8-1.REMと'のメモリ効率の違い

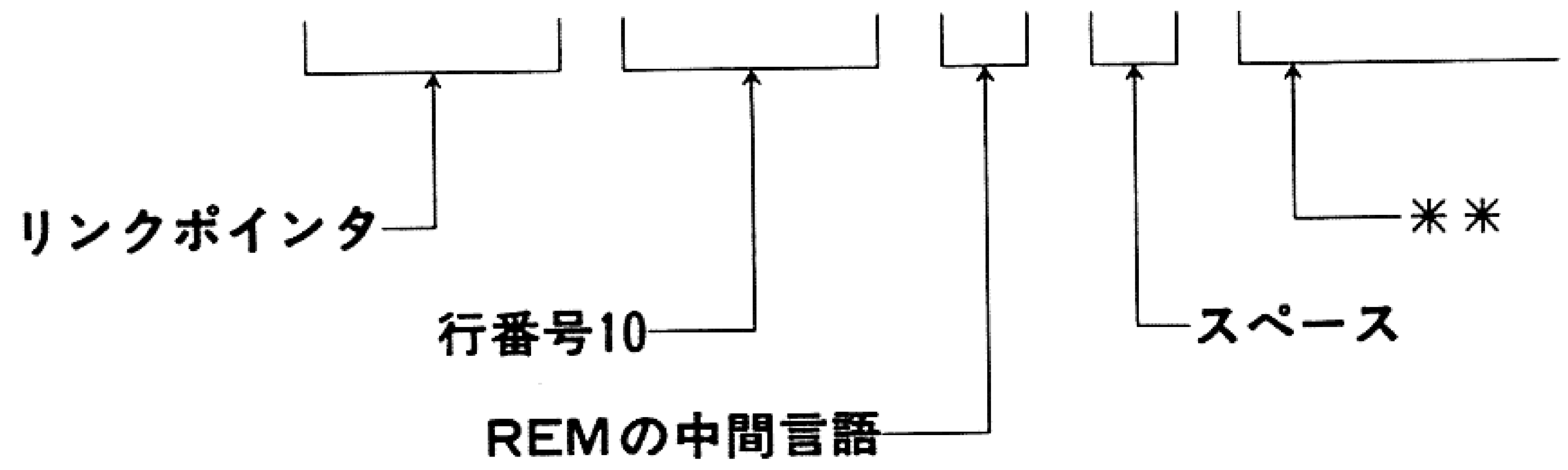
プログラムをわかり易くするために、プログラムの先頭や、途中にREMARK文を設けます。

REMARK文には、〔REM〕または〔'〕を用いますが、メモリへは、異なる形で記憶されます。

(1)REMを使った場合

```
10 REM *****************************
20 REM *                           *
30 REM *       PC-8001    UTILITY  *
40 REM *                           *
50 REM *****************************
```

```
mon
*D8020
8020 00 45 80 0A 00 8F 20 2A 2A 2A 2A 2A 2A 2A 2A 2A
*
```



(2)'を使った場合

```
10 ' *****************************
20 ' *                           *
30 ' *       PC-8001   UTILITY   *
40 ' *                           *
50 ' *****************************
```

```
mon
*D8020
8020 00 47 80 0A 00 3A 8F E4 20 2A 2A 2A 2A 2A 2A 2A
*
```

マルチステートメントの区切り記号"：" ― 3A

REMの中間言語 ― 8F

〔'〕の中間言語 ― E4

すなわち、メモリ内では、

REMの場合………8 F

の場合………3 A・8 F・E 4

〔：〕 ← 3 A　　〔'〕 ← E 4

となっており、〔REM〕が1バイトで済むのに、〔'〕の場合3バイトも使用していることになり、メモリ効率が悪いことになります。

もっとも、REMARK文をつけることそのものが、メモリを余計に使うことになるのでたった2バイトの差がどういうことは、ないのですが………

なお、〔'〕は、マルチステートメントの区切り記号である〔：〕を含んでいるので、〔：〕なしで、行の終わりなどにREM文としてくっつけることができます。

例

```
100 PRINT"ゴウケイ";G        'G...1月 カラ 12月 マデノゴウケイ
```

8-2. REM文の変わった置き方

REM文は、

1. 何のプログラムであるかを示すため、プログラムの先頭におく。

```
10 '***********************************
20 '*                                 *
30 '*         File Name Sorting       *
40 '*                                 *
50 '***********************************
```

2. プログラムの区切りやサブルーチンの初めにおく。

```
 :
170 '      ■■■■■  Sort  ■■■■■
 :
 :
220 REM *****  Print Subroutine  *****
 :
```

などがふつうの使い方ですが、次のような場所に置くこともできます。

3．GOTO文、GOSUB文、THEN文の後には、〔'〕や〔:REM〕をつけずにREM 文を置くことができます。

```
150 F$(N)=FD$(I):GOSUB 220 ...PRINT SUB
150 F$(N)=FD$(I):GOSUB 220 ' Print Sub
```

→ ":REM"や"'" をつけると、小文字のREM文がつけられますが、"REM"、"'" がないと大文字に変換されます。

4．RETURNの後につけるときは、〔:REM〕や〔'〕が必要です。

```
260 RETURN 'To Main Routine
```

5．DATA文の場合は、〔:REM〕または〔'〕をつけて、マルチステートメント化する必要があります。

```
100 DATA 251,280,247,260 ' 9/30, 1981
```

または、

```
100 DATA 251,280,247,260 :REM 9/30, 1981
```

6．一般のステートメントについても、その後に、〔:REM〕または〔'〕をつけて、REM文をつけることができます。

```
100 LPRINT CHR$(27)"E"              'Elite Mode
```

または、

```
100 LPRINT CHR$(27)"E"              :REM Elite Mode
```

REM文のあとに、マルチステートメントで命令を入れても、実行されません。

```
100 REM ******* EDIT ******* :input "Code No. ";c
```

（:input "Code No. ";c）この部分は、実行されません。REM文の一部とみなされます。

8-3.行番号の不思議

行番号は小さい順にリストされることになっています。

```
10  REM ****************************
20  REM *                          *
30  REM *     PC-8001  UTILITY     *
40  REM *                          *
50  REM ****************************
```

ところがあれっ？どうしたのでしょう。

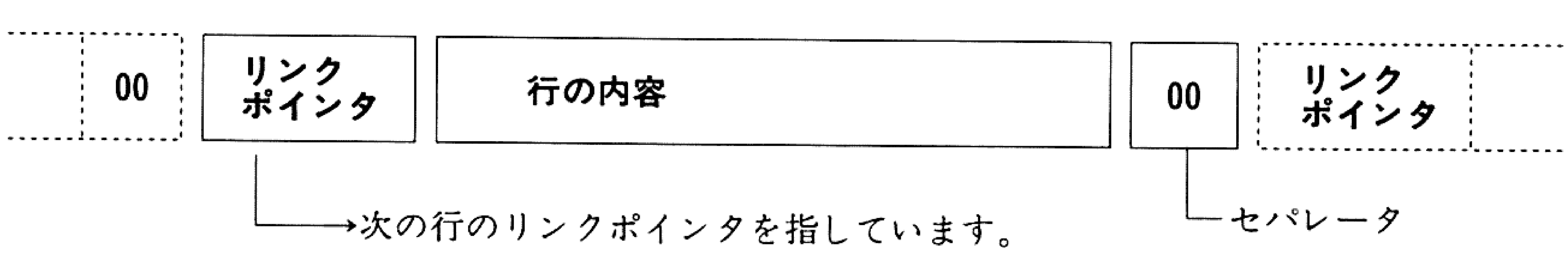
```
list
50  REM ****************************
40  REM *                          *
30  REM *     PC-8001  UTILITY     *
20  REM *                          *
10  REM ****************************
Ok
```

大きい方からリストされましたよ！

```
list
0  REM ****************************
0  REM *                          *
0  REM *     PC-8001  UTILITY     *
0  REM *                          *
0  REM ****************************
Ok
```

あれっ？　こんどは、行番号が全部0になりましたよ。

これらは、メモリ内の行番号の部分を書き換えれば簡単にできるのです。メモリの1行の形式は、



となっているので、このうちの、行番号(行の内容のうちの、最初の2バイト)をモニタモードなどで書き換えれば、良い訳です。

```
mon                                   ┌─行番号10を示す。
*D8020,8100                           ↓
8020 00 44 80 0A 00 8F 20 2A 2A 2A 2A 2A 2A 2A 2A 2A
8030 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A
8040 2A 2A 2A 00 67 80 14 00 8F 20 2A 20 20 20 20 20
 :
 :                                └──→行番号20を示す。

*S8023
8023 0A-00 00-
*S8046
8046 14-00 00-
*S8069
8069 1E-00 00-
*S808C
808C 28-00 00-
*S80AF
80AF 32-00
80B0 00-
```

RENUMすることにより元に戻ります。

8-4.エラーメッセージをすべて表示するには

エラーを起こしたときに表示されるエラーメッセージは、文字列データとして、N-BASICに関しては、37F2番地から3A04番地まで、また、Disk BASICに関しては、D3E5番地からD57E番地に格納されていて、各々、1〜、50〜のエラー番号がつけられています。

ダンプリストでみてみると、

```
run
Start Address    ? 37f0
End   Address    ? 3a04

37F0  27 00 4E 45 58 54 20 77  69 74 68 6F 75 74 20 46  500  '・NEXT w ithout F
3800  4F 52 00 53 79 6E 74 61  78 20 65 72 72 6F 72 00  572  OR・Synta x error・
3810  52 45 54 55 52 4E 20 77  69 74 68 6F 75 74 20 47  57B  RETURN w ithout G
3820  4F 53 55 42 00 4F 75 74  20 6F 66 20 44 41 54 41  4A0  OSUB・Out  of DATA
3830  00 49 6C 6C 65 67 61 6C  20 66 75 6E 63 74 69 6F  5D2  ・Illegal  functio
3840  6E 20 63 61 6C 6C 00 4F  76 65 72 66 6C 6F 77 00  57E  n call・O verflow・
3850  4F 75 74 20 6F 66 20 6D  65 6D 6F 72 79 00 55 6E  5A9  Out of m emory・Un
3860  64 65 66 69 6E 65 64 20  6C 69 6E 65 20 6E 75 6D  607  defined  line num
3870  62 65 72 00 53 75 62 73  63 72 69 70 74 20 6F 75  5FC  ber・Subs cript ou
3880  74 20 6F 66 20 72 61 6E  67 65 00 52 65 64 69 6D  587  t of ran ge・Redim
3890  65 6E 73 69 6F 6E 65 ^C
```

というふうに、アスキーコードで格納されていて、各文字列の間は、00で区分されていることがわかります。

これらを表示するには、次のような簡単なプログラムですみます。

```
list
100 AD=&H37F1:N=1
110 IF AD=&H3A04 THEN AD=&HD3E5:N=50:LPRINT:LPRINT
120 IF AD=&HD57E THEN END
130 AD$=CHR$(PEEK(AD))
140 IF AD$<>CHR$(0) THEN LPRINT AD$;:GOTO 160
150 LPRINT:LPRINT USING "##...";N;:N=N+1
160 AD=AD+1:GOTO 110
Ok
```

N-BASICのみのときは、110行を削除し、120行を110行に変更しアドレスを&H3A04に変更してください。画面に出力するときは、LPRINT を PRINT に変えてください。

実行例

```
 1...NEXT without FOR
 2...Syntax error
 3...RETURN without GOSUB
 4...Out of DATA
 5...Illegal function call
 6...Overflow
 7...Out of memory
 8...Undefined line number
 9...Subscript out of range
10...Redimensioned array
11...Division by zero
12...Illegal direct
13...Type mismatch
14...Out of string space
15...String too long
16...String formula too complex
17...Can't continue
18...Undefined user function
19...No RESUME
20...RESUME without error
21...Unprintable error
22...Missing operand
23...Line buffer overflow
24...Position not on screen
25...Bad File Data
26...Disk BASIC Feature
27...Communications Buffer Overflow
28...Port not initialized
29...Tape read ERROR
```

```
50...FIELD overflow
51...Internal error
52...Bad file number
53...File not found
54...Bad file mode
55...File already open
56...Disk not mounted
57...Disk I/O error
58...File already exists
59...Disk already mounted
60...Disk full
61...Input past end
62...Bad file name
63...Direct statement in file
64...Bad allocation table
65...Bad drive number
66...Bad track/sector
67...Deleted record
68...Rename across disks
69...Sequential after PUT
70...Sequential I/O only
71...File not OPEN
72...File write protected
73...Disk offline
```

8-5.RUN〜とGOTO〜の違い

RUN〜とGOTO〜、同じようで違います。GOTO〜はその行からそのまま実行をするのに対して、RUN〜の場合は、実行する前にすべての変数をクリアし、ファイルをクローズします。したがって、新しくプログラムを実行する場合にはRUNを、途中でSTOPしたプログラムを別の場所から再開するためにはGOTOを使用するのがよいでしょう。

これと同様なものとして、ENDとSTOPの違いがあります。STOP文の場合には、Break in〜が表示され、ブザーが鳴ってからOKが表示されますが、ファイルはOPENされたままです。END文の場合はそのままOKが表示されますが、その前にすべてのファイルがクローズされます。もちろんどちらの場合も変数はそのままです。

8-6. BEEP, MOTORに変数が使える

ご存じですか？　BEEPやMOTORに変数が使えるんです。いや、変数だけでなく式も使えるんです。式の値が０のときBEEP０、０以外の値のときBEEP１と同じになります。MOTORも同じです。ただし、式の値は0～255でなければなりません。

例えば、A＝5：BEEP Aを実行するとブザーが鳴り始めます。

また、IF P＝5 THEN BEEP 1 ELSE BEEP 0は、BEEP－(P＝5)と書けます。

(P＝5)の前に－(マイナス)がつけてあるのは、式の値が0～255でなければならないからです。

8-7. DATE$にうるう年がない

日付を表わすDATE$、これにはうるう年がありません。2月28日の次は3月1日になってしまいます。これというのも、実は、DATE$は、年のくり上がりがないからなのです。PRINT DATE$で表示される年は、単にDATE$＝"○○/○○/○○"で代入したものが、そのまま表示されているだけなのです。

もう一つ、DATE$はおもしろい事ができます。次のものを実行して下さい。

```
AA＝5
DATE$＝"82/AA/01"
PRINT DATE$
```

何と、月の部分には変数が使えるのです。

8-8. DMAをストップさせて実行速度アップ

PCでは画面表示のためにDMAといって、DMAコントローラが、CPUを止めて、VRAMを読んで、データをCRTコントローラへ送るという作業を行っています。この間はプログラムの実行は止まっています。

したがって、DMAを止めてやれば、プログラムの実行が30%近く速くなります。DMAを止めるためには、「OUT104,0」を実行します。これを実行すると画面が消えてしまいます。DMAが行なわれなくなったからです。［CTRL］＋Gを押して下さい。ブザーが鳴りますが、時間が短かくなったでしょう。

DMAをもどすためには「WIDTH,」を実行します。DMAがもどると、また画面が表示されます。次のサンプル・プログラムで行番号10を入れたのと入れないのとで実行速度を比べてみてください。

```
10 OUT104,0
20 TIME$="00:00:00"
30 FOR I=1 TO 5000
40 S=S+I
50 NEXT
60 WIDTH80
70 PRINT S
80 PRINT TIME$
```

8-9.画面リバース

たとえばビーム砲が爆発したとき、画面を一度にリバースさせられたら…と思ったことはありませんか？また実際にそれを行なっているプログラムも見うけられます。

これは実に簡単で、OUT 81，33を実行するだけです。画面が反転し、黒ぬきの文字が表示されます。もとにもどすにはOUT 81，32を実行します。

8-10.グラフィックスイッチ

グラフィックスイッチを立てるとグラフィックモードになるというのはどういうことでしょうか。まず、第1にグラフィックスイッチを立てただけではグラフィックモードにはならないということです。COLOR,,1を実行したあと、PRINT CHR$(12)を行なわなければなりません(グラフィック→キャラクタの時はCOLOR 0,,1 をしないといけないときがあります。)

グラフィックモードとは、グラフィックを多く使用することを仮定したモードです。PC－8001では1行中で20回しかアトリビュートを変化させることができません。キャラクタモードでドットを打つと、ドットの前後でキャラクタ→グラフィック→キャラクタと2回のアトリビュートの変化が起こっています。したがって、数多くのドットを打って行くとすぐに限界に達してしまいます。

グラフィックを使用していて、画面に変なキャラクタが表示された経験はあるでしょう。これは、アトリビュートの変化が20回の限界を超えてしまい、もはやグラフィックを表示することができなくなり、対応するキャラクタが表示されてしまったのです。グラフィックモードで使用すると、このようなことは起こりにくくなります。(キャラクタを多用しない限り起こりません)。

逆にグラフィックモードでキャラクタを使っていると、キャラクタがグラフィックに化けることがあります。これも同じ原因です。

つまり、グラフィックモードとはグラフィックがキャラクタに化けることが少ないモードであり、キャラクタモードとはキャラクタがグラフィックに化けることが少ないモードです。

別の言い方をすれば、グラフィックモードはグラフィックのみを使用している限り「化け」が出ないモードで、キャラクタモードはその逆ということになります。

最後に、次のプログラムをキャラクタモードとグラフィックモードの両方で実行して、「化け」の頻度の違いを十分見て下さい。画面中、四角いベタが出た場所は、キャラクタ→グラフィックの化けが起こった所で、@やETBSなど"w"でない文字が出た場所はグラフィック→キャラクタの化けが起こった場所です。

```
10 WIDTH 80,25:CONSOLE0,25,0,0:COLOR,,1:PRINT CHR$(12)
20 X=RND(1)*160:Y=RND(1)*100:PSET(X,Y)
30 X=RND(1)*79:Y=RND(1)*24:LOCATEX,Y:PRINT"w"
40 GOTO 20
```

8-11. CHR$(0)とヌルストリング("")のちがい

CHR$(0)と""は、同じものと、錯覚しがちですが、その文字列の長さが、CHR$(0)は1、""は0という根本的な違いがあります。

実際に文字変数に代入して、メモリへの格納状態を見てみるとよくわかります。

```
a$=chr$(0)
Ok
b$=""
Ok
print hex$(varptr(a$)),hex$(varptr(b$))
8026          802C
Ok
mon
*D8020
8020 00 00 00 03 00 41 [01] [1A CF] 03 00 42 [00] [1A CF] 0A
*
*
*
*
*DCF10
CF10 46 00 4E 45 50 4F 38 30 32 43 [00] 00 00 FF 00 00
*
```

VARPTR関数でストリングディスクリプタのアドレスを調べ、monモードでそこを見ると各々文字数が1、0であることがわかり、文字列格納先が同じCF1Aとなっていて、そこの内容は00であるので、CHR$(0)は、16進数00で格納されていて、""は、格納先アドレスはあるがそこには文字列が存在しないことになります。

8-12.1行は、256文字以上可能である!?

マニュアルには、〝1行は、行番号も含めて255文字以内である。〟とありますが、正確には、次のようなことが言えます。

①画面上で入力できる文字数は、行番号も含めて、最大254文字である。

確認するために、先ず、適当に255文字以上の行をつくって、listしてみてください。

```
100 ?"1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234
567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234
567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234
567890 12345678|90 1234567890 1234567890 1234567890 1234567890 1234567890"
              └── listで表示される範囲(行番号も含めて254文字)
list 100
100 PRINT"1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 12345678
OK
```

listして表示されたのは、KEY入力時の画面で254文字目までです。

次に、KEY入力時、〝?〟と省略形を用いずに〝print〟で入力して、同様にlistすると、

```
100 print"1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234|567890 1234567890 1234567890 1234567890 1234567890 1234567890"
              └── listで表示される範囲(行番号も含めて254文字)
list 100
100 PRINT"1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
1234567890 1234
OK
```

となり、今度も254文字までが受け入れられて表示されます。すなわち、省略形〝?〟を用いた時の方が、listで表示される文字数は、多くなります。ところが、list後にその行へカーソルを移動して、[RET]キーを押すと、254文字を越える部分は、削除されてしまいます。

キー入力バッファは、EC96～ED93番地に254バイト分だけあり、この容量により制限を受けているものと思われます。

②listで表示される1行は、メモリ上で、リンクポインタも含めて255バイトまでである。

前述の100行のメモリ上の内容をモニタモードで見てみましょう。

```
list
100 PRINT"1234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
12345678901234
Ok

mon
*D8020,813F
8020 00 1C 81 64 00 91 22 31 32 33 34 35 36 37 38 39
8030 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35
  :
8110 34 35 36 37 38 39 30 31 32 33 34 00 00 00 00 00
8120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
```

ここで、811C番地を指しているリンクポインタを少し後方(例えば8130番地)へずらして、その間を0～9の数字を表わすキャラクタコード(30～39)で埋めて、どこまでlist するか見てみます。このとき812F番地は、セパレータとして00を残しておき、8130、31は、プログラムの終わりを示すリンクポインタとして00、00を入れておきます。

```
*S8021
8021 1C-30 81-81 64-
*S811B
811B 00-35 00-36 00-37 00-38 00-39
8120 00-30 00-31 00-32 00-33 00-34 00-35 00-36 00-37
8128 00-38 00-39 00-30 00-31 00-32 00-33 00-34 00-
*
*D8110,813F                                  ┌→ ここから先を書き換えた。
8110 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39
8120 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 00
8130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
Ok

list
100 PRINT"1234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1234567890123456789
Ok            └─ 表示が増えた部分
```

メモリを書き換える前よりも5文字多く表示され、メモリ上では、811F番地までを表示していることになります。すなわち、リンクポインタの先頭番地を含め255バイト(8021H＋FFH－1)を表示する訳です。

ただし、最後の方に、中間言語があると、そのKEYWORDの文字数により、255バイトまで表示されない場合があります。

また、このようにして、PRINT文中の文字を256文字以上にした場合、実行して表示されるのは、最初から256を引いた残りの文字数のみです。

この例では、メモリ上で8027～812E番地まで264文字あり、実行しても〝1 2 3 4 5 6 7 8″としか表示されません。

> ③メモリを直接操作し、1行を256バイト以上とすると、256バイト以降は、list には表示されないが、実行される。

次の例でやってみましょう。

```
list
100 CONSOLE0,25,0,0:WIDTH80,25:PRINT CHR$(12);
110 PRINT"**********************************************************************
***********************************    KEY WORD ｦ INPUT ｼﾃｸﾀﾞｻｲ    ***************
********************************************************************************
**********"
120      :PRINTTAB(35):INPUTK$:IFK$<>"999"THENNEW
130 PRINTCHR$(12):LINE(0,0)-(159,89),PSET,B
140 LOCATE35,11:PRINT"OK !!"
150 GOTO140
OK
```

注）120行の始めの方のスペースは、110行と結合したとき、:PRINTTAB…の前までに255バイト以上確保するためのものです。

このプログラムは、120行で、KEYWORD〝999″が入力されたら、〝OK !!″を画面中央に表示し、〝999″でなかったら、プログラムが消滅する(newされる)という恐ろしい？プログラムです。しかし、listしてみれば、誰にでも簡単にバレてしまいます。

そこで、110行と120行をつないで、1行を256バイト以上になるようにし、120行の肝心な部分が、256バイト以降にくるようにすれば、listしても見えなくなるはずです。

そのためには、

①120行のリンクポインタを指している110行のリンクポインタを130行のリンクポインタの先頭アドレスに変更する。

②110行と120行の間を、スペースコード(20H)などで埋める。(セパレータ〝00″からマルチステートメントの区切り記号〝：″(3AH)の前まで)

という操作が必要です。

上記プログラムのメモリ内容を見て実際にやってみましょう。

```
mon
*D8020,815F
8020 00 [40 80] 64 00 9F 11 2C 0F 19 2C 11 2C 11 3A A0
8030 0F 50 2C 0F 19 3A 91 20 FF 96 28 0F 0C 29 3B 00
8040 [38 81] 6E 00 91 22 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A
  ・    ↑
  ・    └── 110行のリンクポインタ
  ・                                        ┌── 120行のリンクポインタ
                                          ↓
8130 2A 2A 2A 2A 2A 2A 22 00 [57 81] 78 00 20 20 20 3A
8140 91 D9 0F 23 29 3A 85 4B 24 3A 8B 4B 24 F2 F0 22
8150 39 39 39 22 D8 94 00 [76 81] 82 00 91 FF 96 28 0F
*                             ↑
                              └── 130行のリンクポインタ
OK
```

先ず、110行のリンクポインタ（8040､41番地）に､130行のリンクポインタの先頭アドレス(8157H）を書き込みます。

```
mon
*S8040
8040 38-57 81-81 6E-
*
```

次に、110行と、120行の間をスペースコード(20H)などで埋めます。

```
*S8137
8137 00-20
8138 57-20 81-20 78-20 00-20 20-
*
Ok
```

Basicモードに戻って、listしてみると、

```
list
100 CONSOLE0,25,0,0:WIDTH80,25:PRINT CHR$(12);
110 PRINT"**********************************************************************
***********************************   KEY WORD ｦ INPUT ｼﾃｸﾀﾞｻｲ   ***************
********************************************************************************
**********"
130 PRINTCHR$(12):LINE(0,0)-(159,89),PSET,B
140 LOCATE35,11:PRINT"OK !!"
150 GOTO140
Ok
```

120行は表示されません。

では、実行してみましょう。どうでしょう！隠された部分も、ちゃんと実行されていることがわかります。

もちろん、テープやディスクにSAVEしても、その内容は、保存されて、LOAD､RUN可能です。

1つ注意しなければならないことは、この110行にカーソルを移動させ､RET キーを押すと、255文字以降は、削除されてしまうことです。

8-13.数値は､メモリ内でどう表現されているか

数値の型には、次の3つがあります。

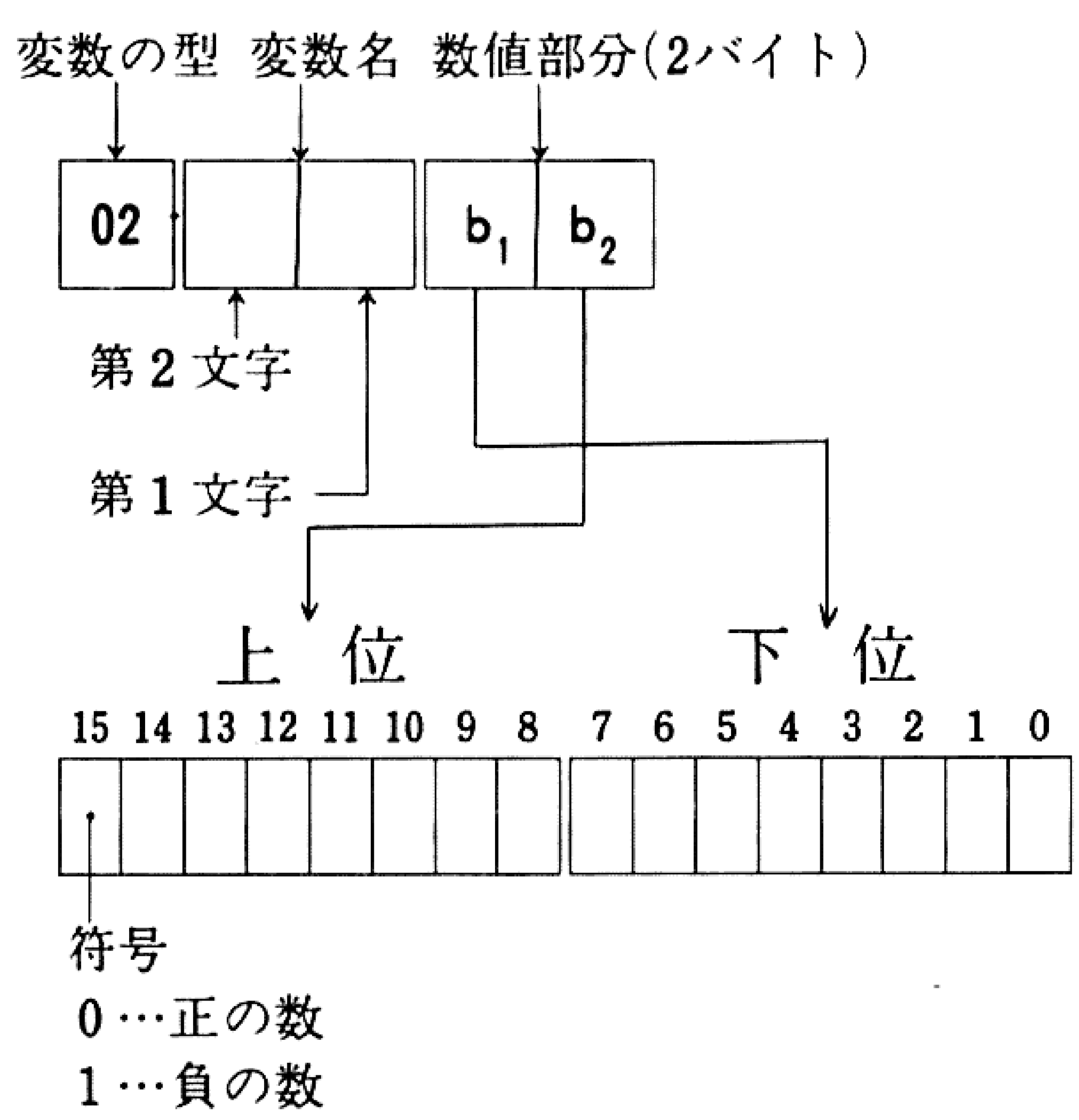
〔図表8-1〕

数値の型			数値部分使用メモリ数	記号
整数型		−32768～32767の整数	2バイト	%
実数型	単精度型	$\pm 1.70141 \times 10^{38} \sim \pm 2.93874 \times 10^{-38}$	4バイト(有効数字6桁)	!(省略可)
	倍精度型		8バイト(有効数字16桁)	#

数値変数名の後に％！＃をつけて区別します。何もつけない場合は、単精度型として処理されます。(ただし型宣言が実行されていない場合に限ります。)

(1) 整数型………2バイト

〔図表8-2〕



7FFF……	32767
7FFE……	32766
⋮	
0001	1
0000	0
FFFF	−1
⋮	
8001	−32767
8000	−32768

$n = \&Hb_2b_1 = (b_2) * 256 + b_1 \longleftrightarrow b_2 = n¥256, \quad b_1 = n \bmod 256$

(例)

```
ab%=12345
Ok
print hex$(varptr(ab%))
8026
Ok
mon
*D8020
8020 00 00 00 |02|42|41|39|30| FF FF FF FF FF FF FF FF
*
```

VARPTRは、数値部分を指しています。

```
print &h3039
 12345
Ok
```

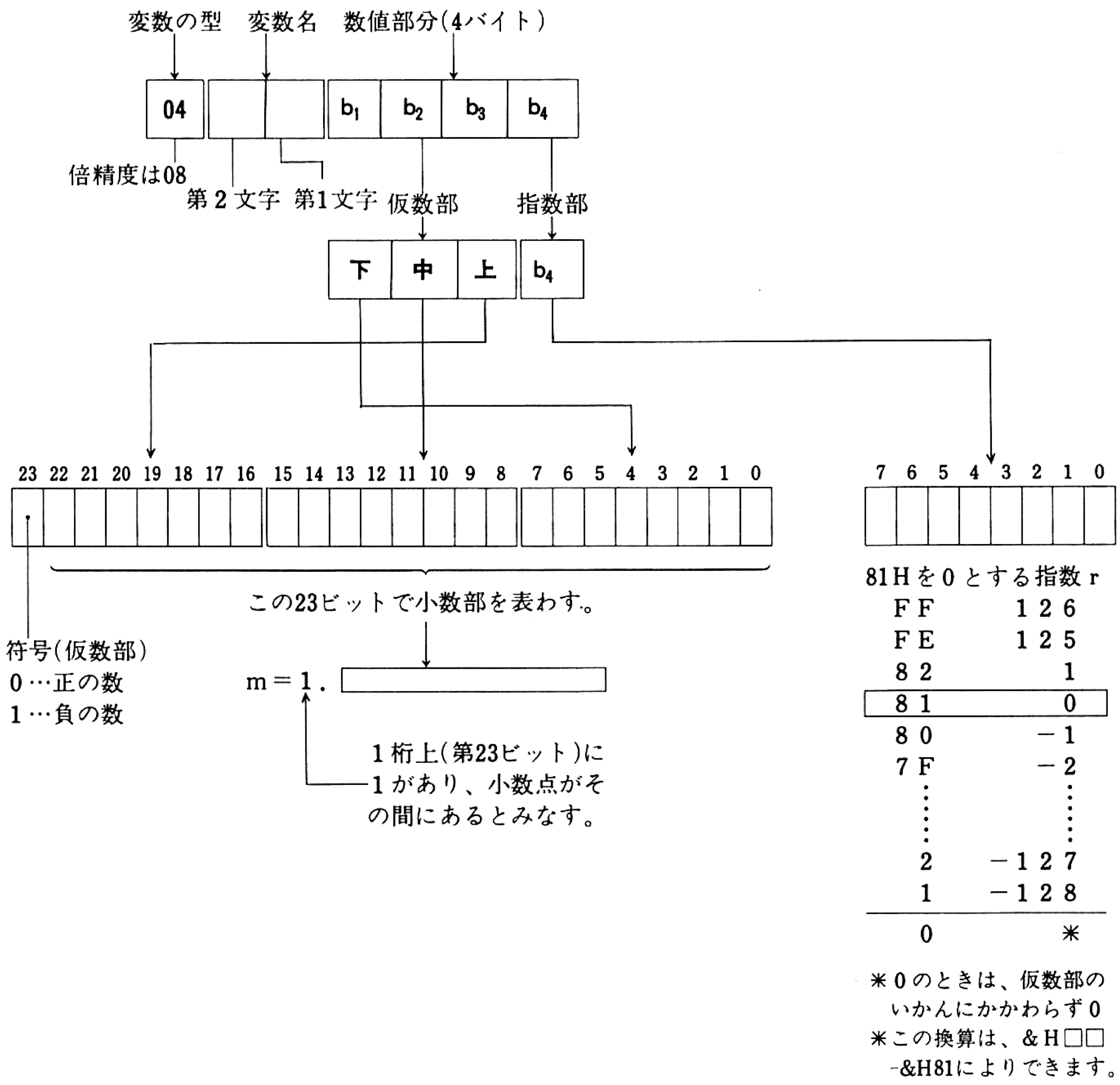
10進数に直すと、12345となります。

(2) 実数型

実数型は、単精度型、倍精度型とも、浮動小数点表記法により表現されています。

単精度変数を例にして、説明します。

〔図表8-3〕



数値nは、

$$n = \pm m \times 2^{r}$$

↑仮数部最上位ビットが0のとき＋、1のとき－

で表わされます。

具体的に次の例で見てみましょう。

```
cd=123.456
OK
print hex$(varptr(cd))
8026
OK
mon
*D8020
8020 00 00 00 |04|44 43|78 E9 76 87| FF FF FF FF FF FF
*                 D  C
                            └──指数部87H→6(87H−81H)
```

|0|1 1 1 0 1 1 0|1 1 1 0 1 0 0 1|0 1 1 1 1 0 0 0|

↓

$+1.11101101110100101111000\times 2^{6}$

2^{6}ということは、仮数部の小数点を右へ6桁移動することを意味します。したがって書き換えると、

```
  7      B        1    D      2      F
|1 1 1|1 0 1 1| . 0|1|1 1 0 1|0 0 1 0|1 1 1 1|
                  └──────────14桁──────────┘
```

となります。これは、計算すると、

$$7BH+1D2FH/2^{14}\fallingdotseq 123.456$$

となり、最初の代入数値にほぼ等しいことがわかります。
倍精度の場合、仮数部が、7バイトになり有効数字が16桁となるだけです。
表現できる数値は、絶対値が

$$1.111\cdots\cdots\cdots 11\times 2^{(FFH-81H)}=1.70141\times 10^{38}$$
$$1.000\cdots\cdots\cdots 00\times 2^{(1-81H)}\fallingdotseq 2.93874\times 10^{-39}$$

の範囲となります。

8-14. VARPTRについて

VARPTR関数には、いくつもの有用な使い方がありますが、その中から2つ紹介しましょう。

1.画面を文字列変数に

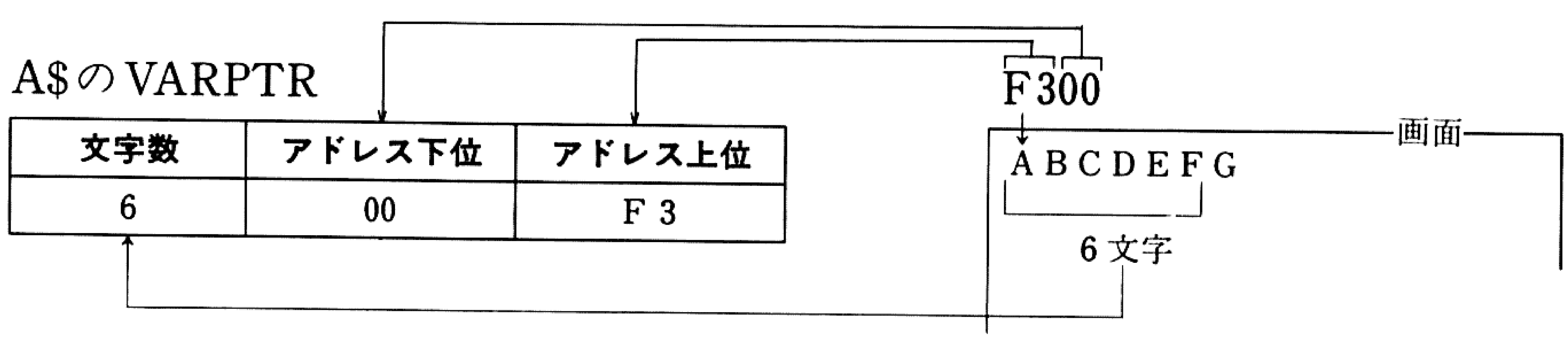
画面上に表示されている文字を文字列変数内にとり込む方法です。数字を読みたい時は一度文字列として読み込んで、VAL関数によって数に変換します。また、この方法は80文字、72文字モードの時にしか使えません。

基本的には、文字列変数のディスクリプタが、画面上を指すようにするわけです。下のプログラムを見て下さい。

```
list
10 WIDTH80:PRINT CHR$(12);"ABCDEFG"
20 A$="":P=VARPTR(A$)
30 POKE P,6:POKE P+1,0:POKE P+2,&HF3
40 A$=A$+"":PRINT A$
OK
```

10行では画面の左上にABCDEFGと表示しています。20行で、A$という、画面上の文字を取り込む変数にヌルストリングを代入し、ディスクリプタを作り、そのアドレスを変数Pにもってきます。(ここでA$に代入する文字列は何でも構いません)

30行が本命です。まず、POKE P,6 によって取り込みたい文字列の長さが6文字ということを指定します。ここではA～Fということです。そうして、P+1、P+2 の2バイトに、文字列の先頭のキャラクタの実アドレスを書き込みます。画面の左上すみのアドレスはF300(72文字モードのときはF304)ですから、POKE P+1,0;POKE P+2,&HF 3となります。



さて、このままでは画面から文字が消去されると、A$の内容も失なわれてしまいます。そこで、行40で、A$＝A$＋"" と行なうことにより、"ABCDEF" を文字領域にコピーしておきます。こうすると"ABCDEF"は完全にA$のものになり、画面には依存しなくなります。

8-15.VARPTRで¥256、mod256を

2.アドレスの上位と下位を別々に

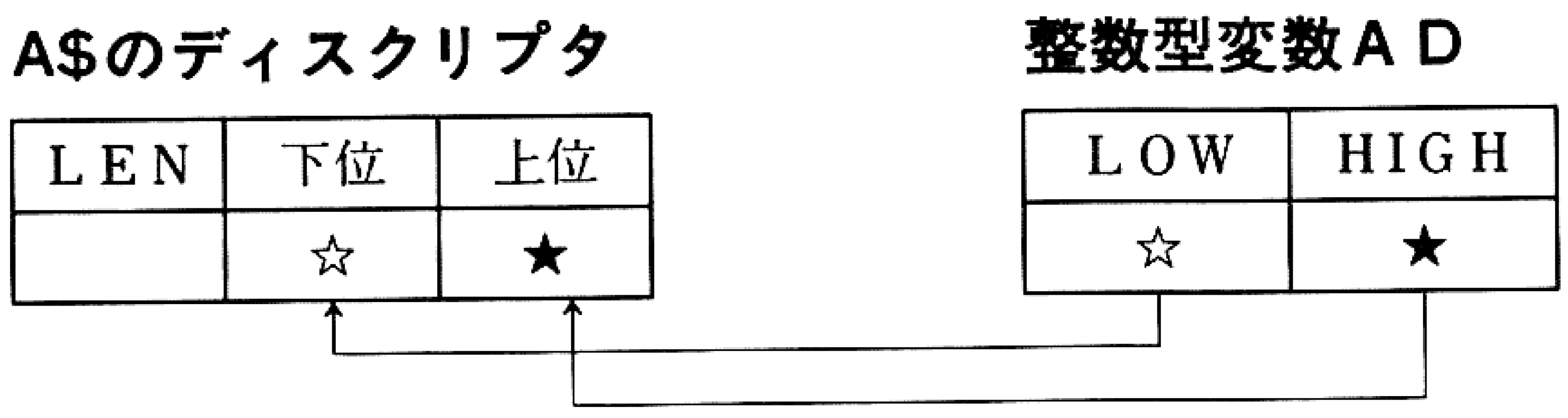
たとえば今の、画面を文字列変数にという場合、ディスクリプタに文字列のアドレスをPOKEしました。例ではアドレスがF300と決まってましたからよいのですが、これが（アドレスが）変数にはいっていた場合はどうでしょう。POKE P+1，AD mod 256：POKE P+2，AD ¥256とやりたいところですね。（変数ADにアドレスが入っているとします） ところがそうはいきません。

もしADが整数型だったとしましょう。このとき、VRAMのアドレスなど8000_{16}～$FF\,FF_{16}$は負の数で表現されます。（例：PRINT & HF300）したがって、AD mod 256、AD ¥256はADの下位バイト、上位バイトを表わしません。

それでは、というのでADを単整度にします。そうすればF300 $_{16}$も正の数で表わせます。ところが、正の数にすると、32767を超えてしまい、modや¥が使えなくなります。そこでしかたなく、INTなどを使って、誤差を気にしながら（実際には誤差は出ませんが、やはり気になります）modや¥の代わりをする、ということになります。

それではVARPTRを使った方法とは何でしょうか。その前になぜ256で割った商とか余りとかを必要としたのでしょうか。それはつまりアドレスの上位バイトと下位バイトの値が別々に欲しかったからです。それならその値を直接もらおうではありませんか！

変数ADを整数型とします。ADは2バイトで表現されており、上位バイト・下位バイトがあります。つまりこれらの値をそのままもらえばよいわけです。



BASICで書くと次のようになります。

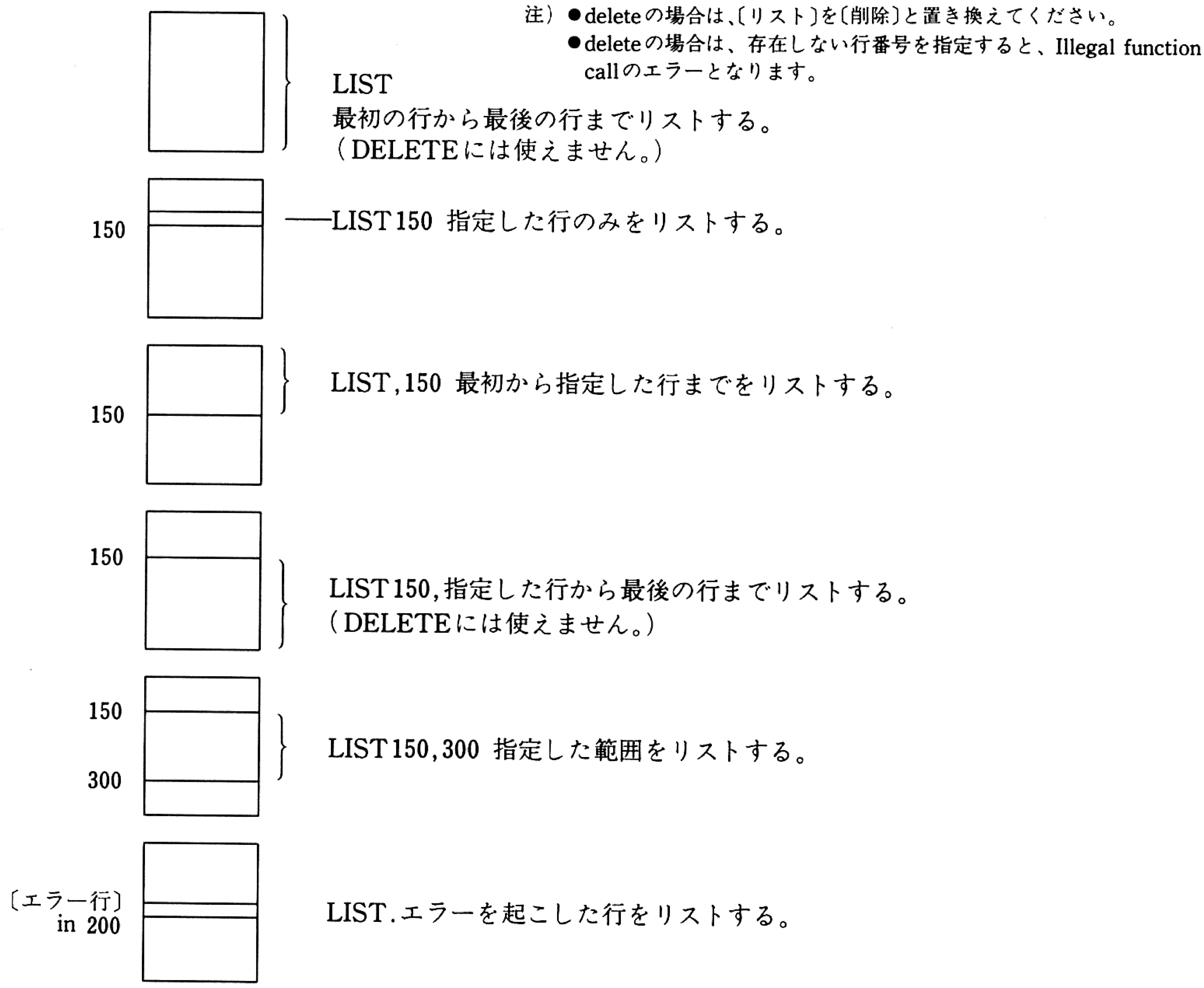
```
POKE P+1、PEEK(VARPTR(AD)):POKE P+2、PEEK(VARPTR(AD)+1)
```

8-16.AUTOのときの似た行の作り方

AUTOを使わない時は同じような行が出てきた場合、カーソルを移動して、行番号を変え、変更ヵ所のみ変更してリターンキーを押すと、それで新しい行ができます。ところがAUTOを使用している場合には同じことをやると、行番号までもがテキストの一部としてとり込まれます。これを解決するには行番号（変更を受ける行の）を消してしまえばよいのです。考えてみればあたり前のことなのですが、意外と知られていないようなので一言。なお老婆心からつけ加えておきますと、AUTOから抜け出さずに以前入力した行をエディットすることは不可能です。

8-17. LISTやDELETEの後に〔.,〕をつけると?

LISTやDELETEには、一般に次の使い方があります。



ここで、最後のLIST.の〔.〕は、

(1)エラーを起こした行番号

(2)LISTコマンドを実行中、表示している最後の行番号。途中でbreak(ストップキーを押して中断)したらその行番号

(3)表示中のリストに対し、修正またはその他の目的でカーソルを移動して RET キーを押した行の番号

(4)AUTO文で現在作成中の行番号
テキスト作成中RETURNキーを押したときのその行番号(AUTO文による作成中も含む)

を意味しています。

この行番号は、EF93，94番地に16進数に変換して格納されます。

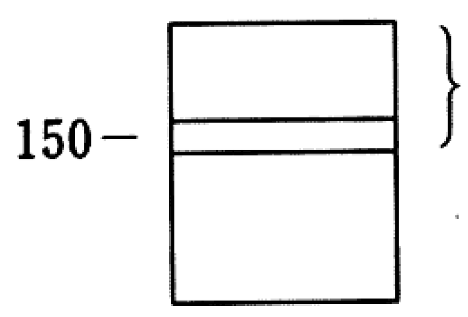
```
mon
¥DEF90
EF90 CD AA 00 32 00 19 82 00 00 00 50 81 78 00 56 81
¥
```

- 56 81 → プログラム中断後、実行再開すべきアドレスを示す。
- 78 00 → プログラム実行中 STOP キー又は、STOPコマンドにより中断した行番号
- 19 82 → エラーを発生した行番号のメモリ上のアドレス8219番地から170行が格納されていることを意味します。
- 32 00 → **〔.〕に相当する行番号**
  00・32を10進数直すと、この例では、50行であることがわかります。
- AA 00 → **前にエラーの発生した行番号**
  00・AAを10進数に直すと170となり、170行でエラーが発生したことを示します。
  エラーが発生した時点では、次のEF93，94番地にも同じ行番号が入ります。

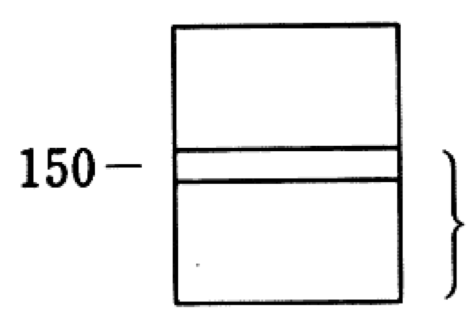
そして、〔.〕はLIST, LLIST, DELETE, AUTOのコマンドで使うことができます。次にその例を示します。

ここで150行とは、エラーが起こった行、リストを中断した行など前述の(1)~(4)に相当する行のことです。

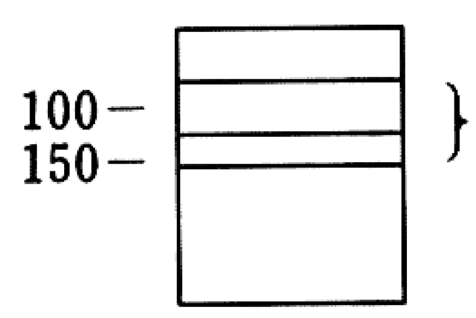
注）DELETEの場合は、〔リスト〕を〔削除〕と置き換えてください。



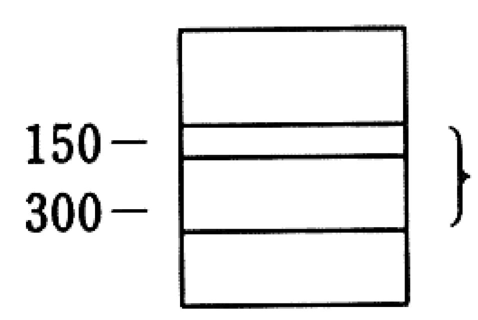
LIST,.　最初の行から150行までリストする。



LIST.,　150行から最終行までリストする。（DELETEには使えません）



LIST100,.　100行から150行までリストする。



LIST.,300　150行から300行までリストする。

AUTO.　150行からAUTOコマンドが実行されます。

LISTコマンドで最終行までリスト後、または最終行より大きい行番号に適当なステートメントをキー入力した後、DELETE400,.を実行すると、400行から最終行まで削除することができます。

8-18. DATA文のランダムアクセス方法

RESTORE文を使わずに、好きなところからのDATAをREADする方法について、1つの例を示します。

1. 区切りとなるDATA文の前に、文字変数の代入文を置く。(代入する文字列は各々、何であってもかまいません。)
2. VARPTRにより、その文字変数の代入文の位置を調べREAD文のポインタ(EFA6, A7) に、その文字列の終わり(〔"〕の次)のアドレスを書込む。

上記方法によるプログラム例と実行例を次に示します。

a～eを入力すると、そのアルファベットを頭文字とするKEYWORDが出力されるというプログラム例です。

```
100 DIM A$(10):GOSUB190
110 INPUT"Character (a-e) ";C$:IF C$<"a" OR C$>"e" THEN 110
120 K=ASC(C$)-&H61
130 AH=0:AL=0:AD=VARPTR(A$(K))+1:AH=PEEK(AD+1):AL=PEEK(AD)+LEN(A$(K))+1
140 IF AL>255 THEN AL=AL-256:AH=AH+1
150 POKE &HEFA6,AL:POKE &HEFA7,AH
160 READ N:FOR I=0 TO N-1:READ K$:PRINT K$,:NEXT
170 PRINT:PRINT:GOTO 110
180 '*******  DATA  *******
190 A$(0)="*":DATA 6,AND,ABS,ATN,ASC,AUTO,ATTR$
200 A$(1)="*":DATA 2,BCD$,BEEP
210 A$(2)="*":DATA 17,CONSOLE,CLOSE,CONT,CLEAR,CLOAD,CSAVE,CSRLIN,CINT
220            DATA CSNG,CDBL,CVI,CVS,CVD,COS,CHR$,CMD,COLOR
230 A$(3)="*":DATA 13,DATA,DIM,DEFSTR,DEFINT,DEFSNG,DEFDBL,DSKO$,DEF
240            DATA DELETE,DSKI$,DSKF,DEC,DATE$
250 A$(4)="*":DATA 9,END,ELSE,ERASE,ERROR,ERL,ERR,EXP,EOF,EQV
260 RETURN
```

```
run
Character (a-e) ? b
BCD$          BEEP

Character (a-e) ? e
END           ELSE          ERASE         ERROR         ERL
ERR           EXP           EOF           EQV

Character (a-e) ? a
AND           ABS           ATN           ASC           AUTO
ATTR$

Character (a-e) ?
```

考慮すべき点

1. DATA文の前におく文字変数の代入文はプログラムの始めに実行しておく。
   そうしないと、ストリングディスクリプタ中に、アドレスが入らない。
   もちろんプログラム中で他の文字列を代入することもいけません。
2. 文字列のアドレスを取込むAL,AHの単純変数の領域は、VARPTR 関数を使用する前に設定しておく。(AL＝0,AH＝0などとして)
   VARPTRで読み込んだアドレスが、AH,ALの単純変数の追加で、配列変数領域が後方にずれて、正しいポインタを指さなくなるからです。

READ文ポインタは、EFA6, A7番地にあり、初期値は、8020H(16KシステムではC020）で、RESTORE文で、例えば、RESTORE210とすると、210行のリンクポインタの直前を指します。

更に、READ nを実行すると、DATA文中の17がnに代入され、READ文のポインタは、次のデータ(CONSOLE)との区切り記号〔, 〕のアドレスを指します。

```
mon
*DEFA0
EFA0 AF 82 AF 82 AF 82 [20 80] 04 04 04 04 04 04 04 04
*
Ok


restore210
Ok


mon
*DEFA0
EFA0 AF 82 AF 82 AF 82 [80 81] 04 04 04 04 04 04 04 04
*
*
*D8180,819F
8180 [00] [C5 81] [D2 00] 41 24 28 13 29 F1 22 2A 22 3A 84
8190 20 31 37 2C 43 4F 4E 53 4F 4C 45 2C 43 4C 4F 53
*

read n:print n
 17
Ok

mon
*DEFA0
EFA0 AF 82 B6 82 B6 82 [93 81] 04 04 04 04 04 04 04 04
*
Ok
```

READ文ポインタ(EFA6, A7)

RESTORE210を実行すると、READ文のポインタが、210行のリンクポインタの直前を指します。

リンクポインタ　行番号210

READ文の実行でREAD文のポインタは、次のデータとの区切り記号である〔,〕のアドレスを指します。

また、VARPTR(A$(2))は、配列変数領域中のストリングディスクリプタ（文字数とポインタがあるところ）を指していて、そのポインタが、テキスト中の文字列を指していてその文字列の終わったところからDATA文が始まっているので、そこのアドレスをREAD文のポインタであるEFA6, A7番地に格納してやれば良いわけです。

これで目的とするDATA文のところにある文字変数を指定するだけで、即座にデータが読めます。

```
print hex$(varptr(a$(2)))
82C4
Ok
```

変数領域中のストリングディスクリプタを指している。

```
mon
*D82C0
82C0  81  01  70  81 [01] [8C  81]  01  09  82  01  77  82  00  00
*
*
*D8180,819F
8180  00  C5  81  D2  00  41  24  28  13  29  F1  22 [2A]  22 [3A]  84
8190  20  31  37  2C  43  4F  4E  53  4F  4C  45  2C  43  4C  4F  53
*
```

テキスト中の文字列を指している。

〔：〕

ここのアドレス818Eを(EFA6, A7)に書込む

もちろん、RENUMも自由にできます。

8-19.動的なRESTORE文

たくさんのDATA文があって、たとえばある変数の値によって、別のDATA文からREADしたいということがよくあります。

そのための方法をいくつか紹介しましょう。

その1：読み飛ばす

必要なデータが出てくるまでデータを空読みし続けます。データ文中のi番目の要素が必要な時は次のようにします。

FOR J ＝1 TO I：READ DT：NEXT

最も一般的な方法ですが、実行時間がかかるのが欠点です。

その2：RESTORE文にON～GOSUBで飛ばす

RESTORE文をそれぞれのDATAに対してつけ、ON～GOSUBによって、必要な行に飛ばします。

```
100 ON I GOSUB 200,210,230:READ DT
200 RESTORE 200:RETURN:DATA 1
210 RESTORE 210:RETURN:DATA 2
230 RESTORE 220:RETURN:DATA 3
```

ON～GOSUBとRESTORE〈行番号〉を組み合わせたうまいやり方ですが、メモリが多く必要な上に、データ数が多くなると見にくいプログラムになってしまいます。

その3：ROM中のルーチンを利用する。

ROM中のルーチンを利用するということは、機械語を使用することです。機械語といってもほんの数バイトです。ここでは使用法のみ述べておきます。

```
100 FOR I=&HFFF0 TO &HFFF5
110 READ D$:POKE I,VAL("&H"+D$):NEXT
120 DATA CD,E1,56,C3,0C,43
130 DEFUSR=&HFFF0
```

このプログラムを実行した後は、A＝USR(行番号)でRESTOREされます。行番号の指定に変数はもちろん、式も使用できるようになります。ただし、RENUMを行って、行番号が変わってしまった場合には、それに応じて式も変えなくてはなりません。

8-20.ERASE利用法

DIMによって配列をとると、各要素の値は0になっています。配列をいろいろ使用していろいろな値が入りました。さて、この配列をクリアしたいのですがどうしますか？このような時には一度配列をERASEしてしまいます。そしてまたDIMするのです。こうすれば簡単に配列の中身をクリアすることができます。

こうすればFOR-NEXTループでクリアするよりも、速くて、バイト数も少なくてすみます。次にDIMA(1000)をクリアするサンプルプログラムを示します。

```
list
10 TIME$="00:00:00"
20 DIM A(1000)
30 ERASE A:DIM A(1000)
60 PRINT TIME$
Ok
run
00:00:00
Ok
```

```
list
10 TIME$="00:00:00"
20 DIM A(1000)
30 FOR I=0 TO 1000
40 A(I)=0
50 NEXT
60 PRINT TIME$
Ok
run
00:00:04
Ok
```

8-21.サブルーチンにCLEAR文はダメ！

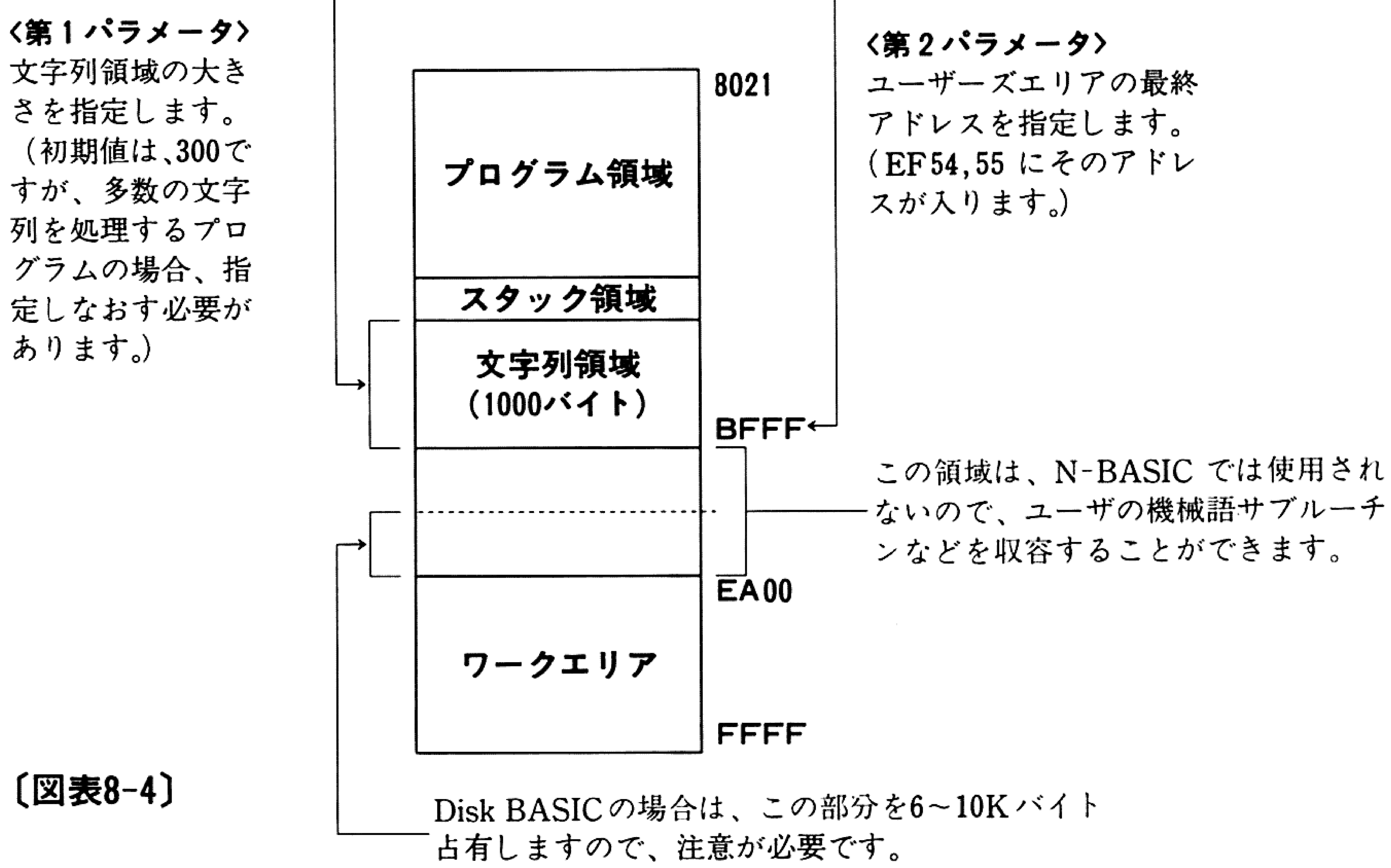
CLEAR文の機能には、次の2つがあります。

(1)CLEAR

すべての数値変数———→0

〃　文字変数———→""…ヌルストリング

(2)CLEAR 1000 , &H BFFF



〔図表8-4〕

CLEAR文の実行により、すべての変数はクリアされ、スタック領域(機械語のpush命令によりレジスタの内容がここに一時保管され、POP命令により再び取り出される。）も初期化されるので、GOSUB文でサブルーチンに飛んでいって、そこでCLEAR文が実行されると、戻る場所がなくなり、エラーとなります。

```
list
110 GOSUB 1000
120 PRINT"FGHIJ"
130 END
1000 ' ■ Subroutine ■
1010 CLEAR 1000,&HBFFF
1020 PRINT "ABCDE";
1030 RETURN
Ok
```

```
run
ABCDE
RETURN without GOSUB in 1030
Ok
```

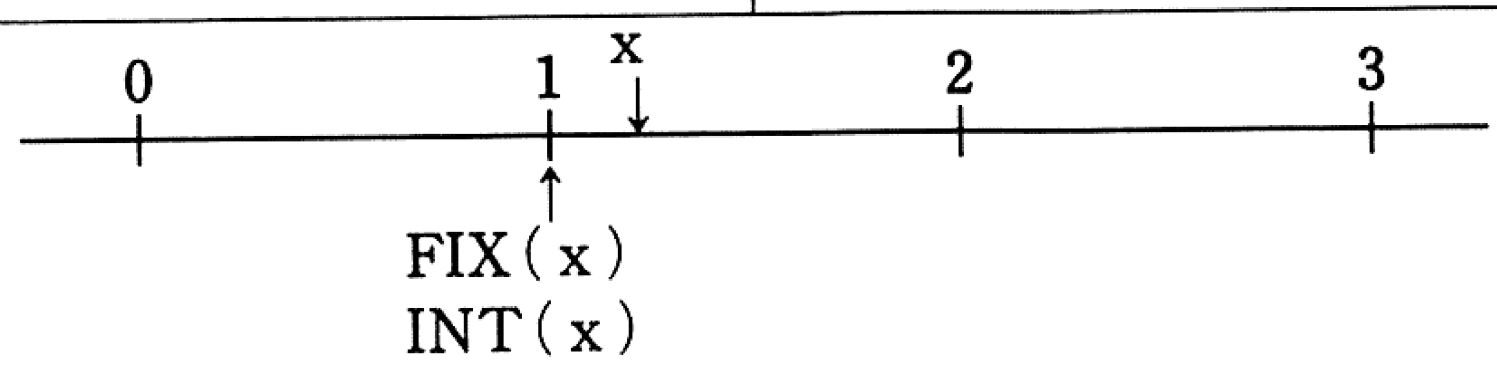
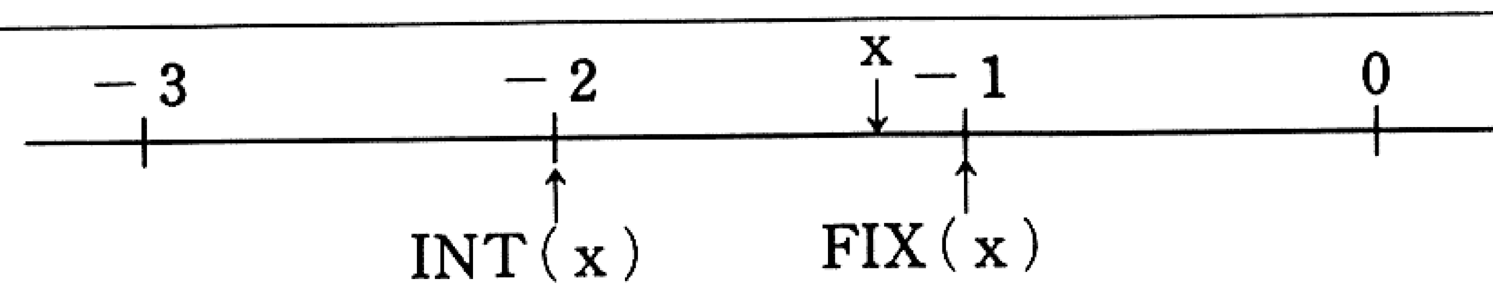
```
list
100 CLEAR 1000,&HBFFF
110 GOSUB 1000
120 PRINT"FGHIJ"
130 END
1000 ' ■ Subroutine ■
1020 PRINT "ABCDE";
1030 RETURN
Ok
```

```
run
ABCDEFGHIJ
Ok
```

CLEAR文は、プログラムの1番最初で、実行するようにしましょう。

8-22.FIXとINTの違いは

〔図表8-5〕

	FIX(x)	INT(x)
機　能	Xの小数点以下を切捨	Xを越えない最大の整数
X≧0 のとき	 0 1 x 2 3 / FIX(x) INT(x) / FIX(x)=INT(x)	
X<0 のとき	 −3 −2 x −1 0 / INT(x) FIX(x) / xが整数でないとき FIX(x)=INT(x)+1 / xが整数のとき FIX(x)=INT(x)	

Xが正の数のときは、同じ値となるが、Xが負の数で小数部分があると異った値となります。

8-23.数値と文字列の変換

〔図表8-6〕

10進数値(整数)		16進数(文字列)	
32767	HEX$() →	7FFF	print hex$(100.8) 64
～		～	
0		0	print &hea00,&hff -5632　　　　255
−1	← &H	FFFF	
～		～	a$="1a00" print val("&h"+a$) 6656
−32768	←	8000	

VAL("&H"+A$)
└16進数が文字変数の場合

- 32768～65535をHEX$で16進に変換すると、8000～FFFFになります。
- 小数点以下は、切り捨てられます。

10進数値(整数)		8進数(文字列)	
32767	OCT$() →	77777	print oct$(7000),oct$(-1) 15530　　　　177777
～		～	
0		0	print &o10000,&o100 4096　　　　64
−1	← &O	177777	
～		～	
−32768	←	100000	

VAL("&O"+A$)
└8進数が文字変数の場合

- 32768～65535をOCT$で8進数に変換すると、100000～177777になります。
- 小数点以下は、切り捨てられます。

アスキーコード(10進)		キャラクタ(文字)	
0	CHR$() →	⋮	print chr$(65),chr$(240) A　　　　　　X
～			
32		"␣"	print asc("H"),asc("年月日") 72　　　　　242
⋮		⋮	
65		"A"	print hex$(asc("0")) 30
247	← ASC()	"秒"	
255		⋮	print chr$(&hb1) ｱ

文字列が2文字以上のとき、先頭のキャラクタのみアスキーコードに変換されます。

10進数値(実数)		数値文字列	
123.45	STR$() →	"␣123.45"	print str$(12.3) 12.3
4096		"␣4096"	print str$(4096) 4096
1.23E−11	← VAL()	"␣1.23E−11"	print val("12.3E-12") 1.23E-11 print val("12/31") 12

数値を文字列に変換すると、正の数の場合、頭にスペースが1文字入り、負の数の場合"−"が入ります。

- 数値以外の文字が出てくると、その前までを数値化します。

8-24.DEFINTで定義された変数は？

変数の型には次の4種類があり、変数名の後にそれぞれ%、!、#、$などの記号をつけてお互いに区別します。

〔図表8-7〕

型 名	記号	定 義 文		使用メモリ
整数型	%	DEFINT	−32768〜32767	数値部2バイト
単精度型	!	DEFSNG	有効数字6桁	〃 4バイト
倍精度型	#	DEFDBL	有効数字16桁	〃 8バイト
文字型	$	DEFSTR	文字列255文字まで	ディスクリプタ3バイト

変数名の後に何もつけていない変数は、N-BASIC起動時は、自動的に、単精度変数となっています。したがって、例えばAとA!、C3とC3!など!をつけた変数と、つけない変数とは、同じものなのです。

しかし、変数の型定義文(DEF…)を実行すると、以降その範囲のアルファベットを頭文字とするすべての変数は、その後に何もつけなくても、定義した型の変数とみなされます。

例えば、DEFSTR A-Eとすると、A,BF,C1,DY,Eなどの変数は、すべて文字変数ということになり、各々A$,BF$,C1$,DY$,E$と同じになります。

言いかえると、DEFINT,DEFSNG,DEFDBL,DEFSTRは、後に型記号(%!#$)のつかない変数を、どの型の変数にするかを定義する文だと言うことになります。

```
a%=123:a!=123.456:a#=123456.7890123456:a$="ABCDE"
Ok
print a
 123.456
Ok
defint a:print a
 123
Ok
defsng a:print a
 123.456
Ok
defdbl a:print a
 123456.7890123456
Ok
defstr a:print a
ABCDE
Ok
```

また、変数Aを文字変数に定義しているとき、単精度変数のAを表示したいときは、

```
Print A !
123.456
```

とすれば良いのです。

同じ数値でも、倍精度（8バイト）よりは、単精度（4バイト）、更には、整数（2バイト）の方が、使用メモリ数が、少ないので、メモリ効率が良く、実行速度も速くなります。

したがって、整数を多く取り扱うプログラムなどでは、

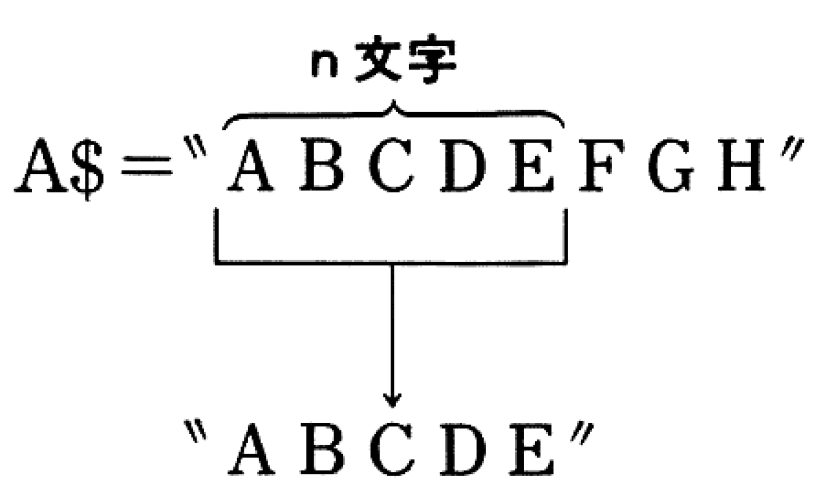
```
DEFINT A－Z
```

として、すべての変数を整数定義しておくと良いでしょう。

8-25.文字列の分解・処理は……

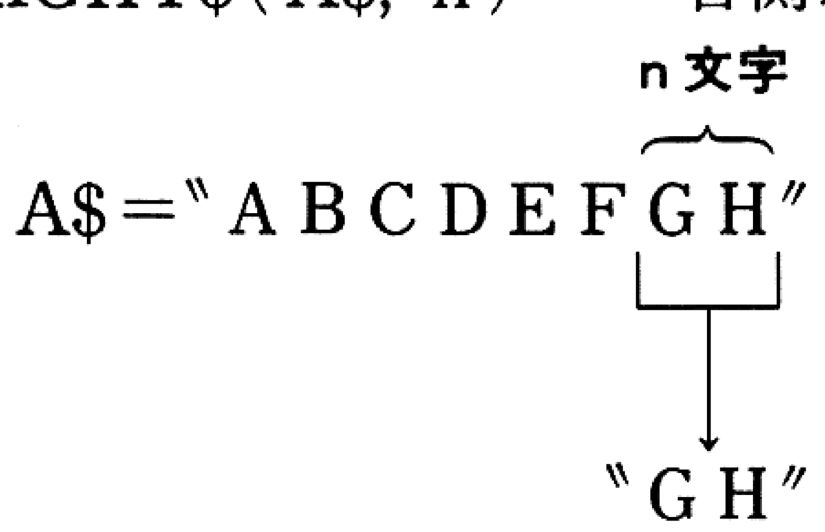
文字列の分解・処理には、LEFT$,RIGHT$,MID$,INSTR,LENなどの関数を用います。

(1)LEFT$（A$, n）………左側からn文字を取出す



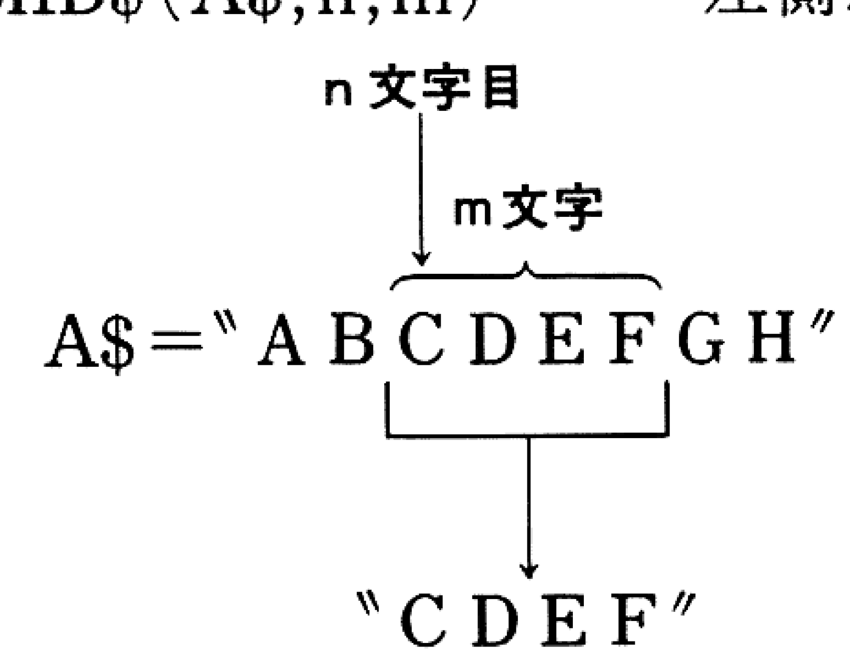
例 ? LEFT$（A$, 5）
ABCDE

(2)RIGHT$（A$, n）……右側からn文字を取出す



例 ? RIGHT$（A$, 2）
GH

(3)MID$（A$,n,m）………左側からn文字目からm文字を取出す



例 ? MID$(A$,3,4)
CDEF

(4)MID$(A$,n)………n文字目以降を取出す。

A$＝〃ABCDEFGH〃

〃DEFGH〃

例？MID$(A$,4)
DEFGH

(5)MID$(A$,n,m)＝B$………n文字目からm文字をB$の頭から
m文字と入れ替える。

n文字目
m文字

A$＝〃ABCDEFGH〃

入れ替える

B$＝〃123456〃

m文字

例B$＝〃123456〃
MID$(A$,5,3)＝B$
？A$
ABCD123H

A$＝〃ABCDEFGH〃

2文字のみ入れ替える

B$＝〃12〃

例B$＝〃12〃
MID$(A$,5,3)＝B$
？A$
ABCD12GH

(6)MID$(A$,n)＝B$………n文字目以降をB$の頭から
A$の文字列長さを越えない
範囲で入れ替える。

n文字目

A$＝〃ABCDEFGH〃

B$＝〃123456〃

例B$＝〃123456〃
MID$(A$,5)＝B$
？A$
ABCD1234

A$＝〃ABCDEFGH〃

B$＝〃12〃

例B$＝〃12〃
MID$(A$,5)＝B$
ABCD12GH

(7)INSTR (A$, B$)………A$中にB$が何文字目から存在するかを調べる。

```
           n文字目
              ↓
A$＝"A B C D E F G H"
              └──┘
               ||
              ┌──┐
     B$＝ "D E"
```

例B$＝"DE"
PRINT INSTR(A$, B$)
4

```
A$＝"A B C D E F G H"
     B$＝"X Y"
```

例B$＝"XY"
PRINT INSTR(A$, B$)
0

例1. IF INSTR(A$, B$) THEN PRINT "アリ" ELSE PRINT "ナシ"

→ないときINSTR (A$, B$)＝0で偽となり"ナシ"とPRINTされる。

例2. IF INSTR (A$, B$)＝4 THEN………

意味→もし、文字列B$がA$の4文字目から存在すれば……

(8)LEN (A$)………文字列の長さを調べる。

```
A$＝"A B C D E F G H"
     └──────┬──────┘
        文字数を調べる
```

例PRINT LEN(A$)
8

付録

付−1　N-BASICインタプリター覧表

付−2　Disk BASICインタプリター覧表

付−3　ワークエリアー覧表

付−4　I/Oポートー覧表

付−5　KEY WORD、中間言語コードと処理ルーチン対応表

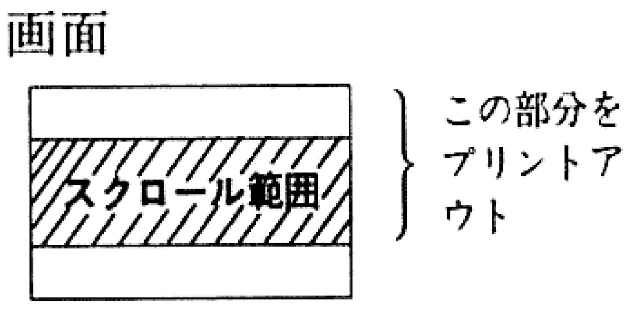
付−6　演算順位表

付−7　アスキーコード表

N-BASIC インタプリター覧表

アドレス	項目名	機能	備考
0000	コールドスタート	RESETボタンと同じ	
0008	ホットスタート	STOP キーを押しながらのRESETと同じ	
0018	1文字出力	EB49に出力機番、ACCに出力コードをセットし、CALLする。またはRST 18H。	出力機番…EB49
00CB	拡張システムチェック	拡張システム(DISKなど)があるとき、NOT ZEROにして戻る。	
00D2	DISK Initialize	DISKの電源を切ったとき、HOT RESETをした時など、ここにJPすると、DISK-BASICが起動する。	
0257	1文字画面表示	ACCに出力する文字コードを入れてCALLすると、対応するキャラクタを表示する。カーソルは、次へ進む。	表示位置X+1：EA64 Y+1：EA63
02D7	1文字画面表示	HにX座標LにY座標を入れてCALLする。	
03A9	カーソル移動	H,LにX+1,Y+1を入れてCALL。	
03D9	VRTCチェック	CRTがバーティカルリトレースになるまで待つ	入力ポート(40H)第5ビット VRTC
03F3	カーソルアドレス	HにX座標、LにY座標を入れてCALLする。HLレジスタに対応するVRAMアドレスが入って戻る。	
0451	ライン消去	DEレジスタに消去する行の先頭アドレスを入れてCALLする。 第0行を消去する例…LD DE,F300H CALL 0451H	
045A	画面クリア	CALL 045AH	
04F8	アトリビュート	HにX座標、LにY座標、ACCにアトリビュートコードを入れてCALLすると、アトリビュートがセットされる。(ただし、該当座標の1キャラクタのみ)	
0664	アドレスサーチ	ACCに画面の行ナンバーを入れてCALL すると、DEにその行の頭のアドレスを入れて戻る。(HLは破壊される)	
06B8	PSET	HLレジスタにデータアドレスをセットし、CALLする。 例、PSET(37,20,2)をしたいとき、データは、28,33,37,2C,32,30,2C,32,29,00	
0705	PRESET	HLレジスタにデータアドレスをセットし、CALLする。(データは、PSETの場合と同様にする。)	

アドレス	項目名	機　　能	備　　考
0792	LOCATE		カーソルスイッチEA5F
07C9	ファンクションキーディスプレイ	ACCに00または05を入れてCALLすると、 00…1～5 05…6～10 }を表示 このACCに入れる数値＋1から5つを表示する。	
0843	WIDTH	HLレジスタにデータアドレスをセットし、CALLする。 例、WIDTH40,25をしたいとき、データは 34,30,2C,32,35,00	
0884	CONSOLE	HLレジスタにデータアドレスをセットし、CALLする。 例、CONSOLE0,25,0,1をしたいとき、 データは、 30,2C,32,35,2C,30,2C,31,00	
08F7	CONSOLE & WIDTH	BCレジスタにFUNCTION KEY表示スイッチ、カラー／白黒フラグをセットしてCALLすると、ワークエリアに従ってセットされる。	F・KEY表示方法…EA5C スクロールの長さ−1…EA5D スクロール開始行+1…EA5E カーソルスイッチ…EA5F F・KEY表示スイッチ…EA60 カラー／白黒…EA61 行数…EA62 カーソルY＋1…EA63 カーソルX＋1…EA64 1行文字数…EA65
0930	WIDTH	EA62に行数、EA65に1行文字数を入れてCALLする。	
093A	WIDTH	Bレジスタに1行文字数、Cレジスタに行数をセットしてCALLする。	
0951	COLOR	HLレジスタにデータアドレスをセットしてCALLする。 例、COLOR3,32,1の場合、データは、 33,2C,33,32,2C,31,00	
09A3	WIDTH （横方向）	ACCに1行の文字数をセットしてCALLする。 40文字……ACC←28H 80文字……ACC←50H	
09D7	WIDTH （縦方向）	ACCに行数をセットしてCALLする。 20行……ACC←14H 25行……ACC←19H	
09F6	DMA ON（25行）	DMAをONにして戻る。（25行になる）	
0A5E	DMA ON（20行）	DMAをONにして戻る。（20行になる）	
0B2E	ファンクションキーOFF	ファンクションキーの表示を消します。 スクロール範囲は、元のままです。	
0B92	グラフィック →カーソル	(H,L)のグラフィック座標をカーソル座標に直す。	

0BD2	カーソル消去	CALLすると、カーソルが消える。	
0BE2	カーソル表示	CALLすると、カーソルが表示される。	
0BF3	カセット入力セット	カセットをONにして入力準備をする。 REMOTE SW-ON状態になる。	
0C2E	MOTOR OFF	CMTをCLOSEにする。 REMOTE SW-OFF状態になる。	
0C3E	MOTOR ON	ポート21HにCE,11を出力し、8251をセットする。	μPD8251…USART
0C49	カセット出力セット	カセットをONにして出力準備をする。 REMOTE SW-ON状態になる。	
0CF1	ストップキーチェック	ストップキーが押されているかどうかをチェックし、押されていれば、キャリーフラグを1にして戻る。 ストップキーが押されていれば、ACCに"3"、ESCキーのときは、"13"が入って戻る。	
0D14	MOTOR OFF	ポート21Hに0E,40を出力し、8251をリセットする。	
0D41	BEEP		
0D5D	MON		
0D60	プリンタへの1文字出力	ACCにASCIIコードをセットし、CALLする。	
0DA1	MOTOR		
0DB8	TERM		
0F75	1文字入力（WAITあり）	キーボードが押されるまで待ち、入力結果をACCにASC IIコードで入れて戻る。	
0F7B	KEY入力チェック	キー入力があったら、キャリーフラグをリセット（CY←0）し、ACCにASC IIコードをセットして戻る。	
0FAC	1文字入力（WAITなし）	キーボードが押されていたら、その入力結果をACCにASC IIコードで入れて戻る。 （押されるまで待ちません。）	
10E5〜117C	キーボード入力ASCII変換（DATA）	キーボードマトリクスをASC IIコードに変換するためのデータ。	
124A	画面プリント	CALLすると、VRAMの内容（画面に表示しているもの）をプリンタに出力する。 第0行から、スクロールウインドウの下端までをプリントアウトする。	画面  LOCATEによる表示の場合、あらかじめ COLOR,&H20 PRINT CHR$(12) 実行しておくと、表示文字間が詰まりません。
1343	KEY		

アドレス	項目名	機　能	備　考
159F	DATE$	画面表示は、52F0	A
15DC	TIME$	画面表示は、52F0	
1602	タイムリード	CALLすると、結果が、ワークエリアのそれぞれのアドレスに入ります。 10進数のまま入ります。15秒→15H 年の部分は、セットしたままのものが入っているだけで、繰り上がりしません。	EA76……秒 EA77……分 EA78……時 EA79……日 EA7A……月 (EA7B……年)
164C	タイムクリア	CALLすると、タイマを初期状態にします。 00：00：00 79／01／01	
1651	タイムリセット	DATE$, TIME$リセット	
1663	タイムセット	EA76～EA7Bに時刻・日付を入れてCALLする。	
1757	初期化ルーチン	コールドスタート時に実行される。	
1838 ∫ 1874	Version メッセージ (DATA)	NEC PC-8001 BASIC……CRLF ↑	
1875	DISK BASIC メッセージ出力	Disk Basic Feature メッセージ出力	
1886 1891	GET PUT		
1B7E	1行入力ルーチン	RETURN, STOPで戻る。	
1B8A	1行入力ルーチン	カーソルのあるところから、1行入力 EC96～ED93(254バイト)にバッファがあり最後の文字の次には、00が入る。	キー入力バッファEC96～ (254バイト)
1BD8	コントロール文字出力		
1EC0	CSAVE		
1F10	CLOAD		
1FBC	SKIP	SKIP：ファイル名をプリントして、ロードをつづける。	
1FC9	Badメッセージ (DATA)	"Bad"メッセージ	
1FEF	Found メッセージ (DATA)	"Found：　"メッセージ	
1FF6	Skip メッセージ (DATA)	"Skip：……"メッセージ	

1FFC	Found	Found：……をプリントする。	
20F9	PORT		
2262	INIT		
240F	単精度減算	(Flow Acc)←(レジスタペア)－(Flow Acc)	Flowting point Accのアドレス F0A3 A4 A5 A6 A7 倍精度 (HL) A8 整数 A9 単精度 AA AB
2412	単精度加算	(Flow Acc)←(レジスタペア)＋(Flow Acc)	
2503	LOG		
2541	単精度乗算	(Flow Acc)←(レジスタペア)×(Flow Acc)	
259C	単精度除算	(Flow Acc)←(レジスタペア)÷(Flow Acc)	
2671	ABS		
2686	SGN		
2689	整数化	Acc(－128～127)を整数にしてFlow Accにストア	
270C	単精度比較		
2739	整数比較		
2778	倍精度比較		
277F	CINT		
27B3	CSNG		
27DF	CDBL		
27FA	Flow Acc チェック	Flow Accがストリングでなかったらエラー	
282C	FIX		
283F	INT		
28D2	整数減算		
28DD	整数加算		
28FD	整数乗算		整数除算…4BEA
29C3	倍精度減算		
29CA	倍精度加算		
2A2B	倍精度の正規化と符号付け		
2AF4	倍精度除算		
2BBE	型判断とポインタセット		
2D13	HLをプリント	HL(0～65535)	
2D22	型変換	数値→文字列	

アドレス	項目名	機能	備考
309F	HEX→DEC (型変換)	F0A8, A9(Flow Acc)の16進数を5桁の10進数の文字列にして、(HL)~(HL+4)に入れて、(HL+5)に00を入れる。	
3148 ~ 3151	DATA (HEX→DEC)	2710……10000 03E8………1000 0064…………100 000A……………10 0001………………1	
31A1	SQR		
31F3	EXP		
3283	RND		
32F6	COS		
32FC	SIN		
335D	TAN		
3372	ATN		
33BD ~ 3466	中間言語処理 (DATA)	中間言語に対応した処理ルーチン表 81H→END ~ D5H→LOCATE	付ー5参照
3467 ~ 34C0	中間言語処理関数 (DATA)	関数の中間言語に対応した処理ルーチン表 FF·81H→LEFT$ ~ FF·ADH→MKD$	
34C1 ~ 34F4	中間言語 Index (DATA)	キーワードの頭文字(ABC……Z)とキーワード格納アドレスの対応表	
34F5 ~ 37A7	キーワード (DATA)	キーワードと中間言語コードの対応表 ABC順	
37A8 ~ 37BC	特殊記号 (DATA)	特殊記号と中間言語コードの対応表	
37C9 ~ 37D2	型変換ルーチン (DATA)	型変換処理ルーチンアドレス 37C9……CDBL:27DF 37CD……CINT:27FA 37D1……CSNG:27B3	
37D3 ~ 37F0	算術演算 処理ルーチン (DATA)	算術演算処理ルーチンアドレス 37D3~倍精度 37DD~単精度 37E7~整数	

3 7 F 2 ～ 3 A 0 4	エラーメッセージ （DATA）	エラーメッセージの表 3 7 F 2……NEXT without FOR ⋮	
3 A 0 5 ～ 3 B 5 B	初期設定 （DATA）	初期設定のとき、EA00～EB56に転送される。 （詳細は、ワークエリアEA00～参照のこと。）	転送先EA00～EB56
3 B 9 D	OUT OF MEMORY チェック	Out of Memory チェックルーチン Cレジスタに必要なスタックレベル	
3 B D F	Syntax error	Syntax error 処理	
3 B F 1	Over flow	Over flow 処理	
3 B F 9	エラー表示	EレジスタにERRコードを入れてCALLする。処理後、BASICのコマンド待ちになる。	
3 C 9 F	コマンド待ち	コマンド待ちに入るアドレス。	
3 D 6 7	コマンド待ち	DEレジスタペアに、EB54,55の内容をセットして、JPすると、コマンド待ちになる。 （変数はクリアされる）	EB54,55 BASICプログラム開始番地
3 D 7 6	プログラムフィット	EB54,55から入っているBASICプログラムをその番地にフィットさせる。 （end＝00,00）	
3 D E 0	NEW	Basicプログラムを抹消する。 コマンド待ちになる。 （JP 3DE0または、＊G3DE0）	
3 D F 4	RUN	Basicプログラムが実行開始される。 （JP 3DF4または、＊G3DF4）	
3 E 5 C	数値入力サブルーチン	？を表示して、入力待ちとなる。	
4 0 9 5	HL-DEを行ない、	HL-DEを行ない、Zフラグをセットする。 （Accは、破壊される）	
4 0 A 6	1文字出力	AccにASCIIコードをセットし、CALLする。	RST18に対応
4 1 5 9	FOR		
4 1 F A	コントロールB	monモードからBASICに戻る。	
4 3 0 2	RESTORE		
4 3 0 C	RESTORE	DEに行番号をセットして、CALLすると、RESTORE行番号	
4 3 2 A	STOP		
4 3 2 F	END		
4 3 8 3	CONT		
4 3 9 6	TRON		

アドレス	項目名	機能	備考
4397	TROFF		
439C	SWAP		
43DF	ERASE		
445B	DEFSTR		
445E	DEFINT		
4461	DEFSNG		
4464	DEFDBL		
44A5	メッセージ出力	Illegal function call メッセージを出力して、BASICコマンドモードに戻る。	
44C7	DEC→HEX	HLレジスタに文字列の先頭アドレスを入れてCALLすると、16進に変換したものが、DEレジスタに入る。	
44E8	CLEAR		
453D	RUN		
4555	GOSUB		
456D	GOTO (GO TO)		
45A3	RETURN		
45BE	DATA		
45C0	ELSE REM		
45DE	LET		
4642	ON		
468C	RESUME		
46C4	ERROR		
46CF	AUTO		
4702	IF		
473A	LPRINT		
4742	PRINT		
4877	LINE	カラーモード時の1行単位での機能指定をする。HLレジスタにデータアドレスを入れてCALLする。 例、LINE10,1のときデータは、 31,30,2C,31,00	
48AC ⟆ 48BE	エラーメッセージ (DATA)	? Redo from start	

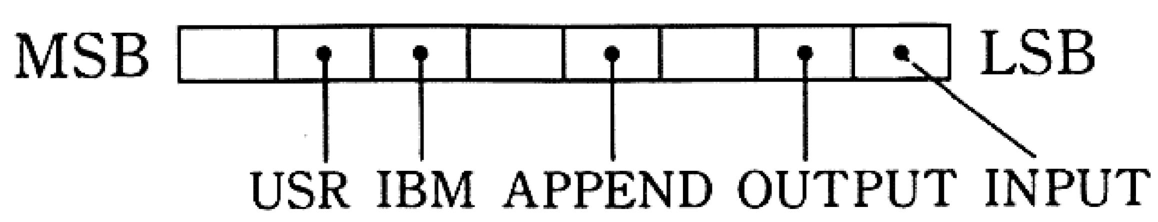
4 8 D A	INPUT		
4 9 3 9	READ		
4 9 D 7	エラーメッセージ (DATA)	? Extra igrnored	
4 A 0 8	NEXT		
4 A 8 F	式の計算	HLがテキストポインタ	
4 B E A	整数除算		
4 C C B	大文字変換	小文字を大文字にする。	
4 E 3 7	DIM		
4 E 3 F	変数サーチ	DEレジスタに変数最下位バイト EF45に型 } が入る。	
5 0 5 1	FRE		
5 0 7 4	LPOS		
5 0 7 9	POS		
5 0 C C	DEF		
5 2 7 0	OCT$		
5 2 7 5	HEX$		
5 2 7 A	STR$		
5 2 7 D	文字列登録	(HL)...データアドレスから00までの文字列をストリングエリアにストアし、Flow Accにに登録する。	
5 2 E D	メッセージ出力	HLレジスタに文字列の先頭を入れてCALLすると、00までの文字列が表示される。 カーソルは、最後に表示した文字の次になる。 (改行はされない。)	
5 2 F 0	画面表示		
5 3 2 F	ガベージ コレクション	(EF79, A)-(EB50, 1) が残りバイト数を示す。	
5 4 5 9	ストリング チェック		
5 4 9 8	ASC		
5 4 A 8	CHR$		
5 4 D F	SPACE$		
5 5 2 9	RIGHT$		
5 5 3 2	MID$		
5 5 5 3	VAL		

アドレス	項 目 名	機 能	備 考
5 6 A 1	INP		
5 6 A D	OUT		
5 6 B 3	WAIT		
5 6 F A	式の評価	0≦式≦255となるべき式を評価してAccに入れる。	
5 7 0 7	LLIST		
5 7 0 C	LIST	BASICのリストを出力し、BASICのコマンドモードに戻る。	
5 8 D 9	DELETE		
5 9 1 1	PEEK		
5 9 1 8	POKE		
5 9 2 A	Flow Acc INT	Flow AccをINTにする。	
5 A E D	RENUM		
5 B 8 5	行番号変換	行番号を行アドレスに変換	
5 B 8 6	行アドレス変換	行アドレスを行番号に変換	
5 B F 0	エラーメッセージ (DATA)	Undefined line number	
5 C 2 C	モニタスタート	PC-8001用モニタの開始番地	5C2Cおよび5C3C番地をCALLした場合は、コントロールBでBASICへ戻れない。
5 C 3 C	モニタスタート	PC-8001用モニタの開始番地	
5 C 5 E	?モニタ	0D,0A,3F(?)を出力してモニタに移る。	
5 C 6 6	*モニタ	"*"を表示してモニタのコマンド受付状態になる。機械語の最後にHALT(76H)の代りにJP5C66を入れておくと、1々RESETする必要がない。	
5 C 6 F ～ 5 C 9 2	モニタ処理ルーチン (DATA)	モニタモードのコマンドに対応した処理ルーチンの対応表	
5 C 9 3	mon→BASIC	モニタからBASICへ戻る。	
5 C 9 9	Sコマンド	セット	
5 D 1 6	Dコマンド	ディスプレー	
5 D 6 8	Gコマンド	ジャンプ	
5 D 7 4	Wコマンド	ライト(カセットへの)	
5 D A E	Lコマンド LVコマンド	ロード(カセットからの)	

5E21	16進入力ルーチン	DEにスタートアドレス、HLにエンドアドレスが入る。	
5E39	HEXチェック	AccにコードをいれてCALLし、16進に使用されるキャラクタかどうか調べる。 16進用キャラクタでなければ、キャリーフラグがセットされる。	
5E4B	バイナリ変換	4桁までの16進ASC IIコードをバイナリコードに変換する。 AccにASC IIコードをセットしてCALL する。	
5E83	16進変換	Accにバイナリコードを入れてCALLする。 結果は、DEレジスタ。	
5EA0	バイナリ変換	DEレジスタにASC IIコードを入れてCALLする。結果は、Accに残る。	
5EBD	アスキー変換 および表示	表示したいメモリの番地をHLレジスタに入れてCALLすると、その番地の内容（バイナリ）を画面のカーソル位置に表示。	
5EC0	アスキー変換 および表示	HLレジスタ中の4桁のバイナリコードをASC IIコードに変換して、画面のカーソル位置から表示する。	
5EC5	BIN→HEX	Accにバイナリコードを入れてCALLすると画面にそのHEXコードを表示する。	
5ED3	HL-DE	HL-DEの演算を行なって、結果として、フラグの内容のみを変える。 Accもこわれない。	
5FAD	キー入力	キー入力、出力サブルーチン （英小文字は、英大文字に変える。03Hでモニタへ）	
5FB0	1文字表示と カーソル表示	Accに文字コードを入れてCALLする。 画面にAccの内容を表示してカーソルも表示	
5FC1	英大文字変換	A～Zまでの小文字を大文字に変換する。 Accに文字コードを入れてCALLすると結果もAccへ	
5FCA	CR & LF	改行する。（CRとLFを出力するサブルーチン）	
5FD4	スペース表示	画面にスペースを表示する。	

DISK BASICインタプリター覧表

アドレス	項　目　名	機　　　能	備　　　考
D3E6 ∫ E9D3	DISK BASIC	DISK BASIC起動時に、システムディスクより、この部分に、データがブロック転送されてくる。	
D3E6 ∫ D57E	エラーメッセージ (DATA)	DISK BASIC用エラーメッセージ表 D3E6…FIELD overflow ⋮	エラーコード…51Hより
D57F	転送ルーチン	ジャンプテーブル用ブロック転送ルーチン D58Bからの135HバイトをF0E1～F215へ転送する。転送後は不用。	
D58B	ジャンプテーブル (DATA)	起動時に、F0E1～F215へ転送される。 転送後は不要。	
D6C0		N-BASICのRUNルーチンより、	
D6CC		ファイルバッファポインタ退避	
D6D3		〃　　　　復活	
D6DA		〃　　　　クリア	
D70A		エラーメッセージポインタ設定 (HL：エラーメッセージの先頭アドレス)	
D737		INPUT＃　メインルーチン	
D774		PRINT＃　メインルーチン	
D804	FORMAT		
D8BF	MOUNT	メインルーチン	
D91E ∫ D940	エラーメッセージ (DATA)	Copies of allocation………	
D941	REMOVE	メインルーチン	
D9C3	LOC		
DA26		Cレジスタ×8，結果はDEレジスタ	
DA90		ファイル名解析	
DB11		ファイルバッファポインタGET (結果はHLレジスタ)	

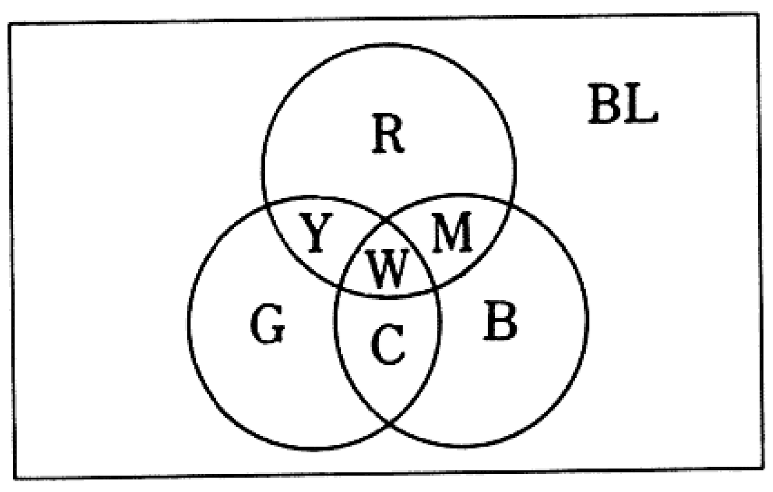
DB3D		FATポインタGET（結果はHLレジスタ）	
DB84	NAME	メインルーチン	
DBC4	ファイル名交換	ファイル名1(E9E2,EA)とファイル名2(E9EB,F3)の入れかえ。	
DBD6	OPEN	メインルーチン Eレジスタ  	
DD2C	ファイルプロテクト		
DDDE	KILL		
DDEB		RUN, LOAD, MERGEメインルーチン	
DDF0	LOAD		
DDF1	MERGE		
DE3D	SAVE	メインルーチン	
E011	RSET		CY＝0
E012	LSET		CY＝1
E0BF	FIELD		
E109	SET		
E15D	ATTR$		
E1D2	MKI$		
E1D5	MKS$		
E1D8	MKD$		
E1EB	CVI		
E1EE	CVS		
E1F1	CVD		
E208	MOUNT	} 初期ルーチン	
E20D	REMOVE		
E240	CLOSE		
E256		IPLで使用。ファイルバッファ、FATエリアイニシャライズ	
E27F	LFILES		
E284	FILES		

アドレス	項目名	機能	備考
E326	PUT #		
E327	GET #		
E3BD		ファイルバッファへ1バイト書く	
E40C		ファイルバッファから1バイト読む	
E44A	DSKI$		
E47C	DSKO$		
E498		DEからHLへCバイトコピー	
E4A2		テキストより3つのパラメータを得る ①, ②, ③ ⇒ ①→A reg ②→B reg ③→C reg	
E812	DSKF		
E823	LOC		
E83B	LOF		
E84C	EOF		
E868	FPOS		
E986	エラー処理	エラーコード設定後エラールーチン(3BF9)へジャンプ	
E9D4	オープンファイル数（DATA）	DISK BASIC起動時、How many files (0〜15)？でインプットした数値(0〜F)が入る。	
E9D5 〜D6	バッファポインタ（DATA）	バッファポインタの始まりアドレス	
E9D7 〜D8	FATポインタ（DATA）	FATポインタの始まりアドレス	
E9D9 〜DA	バッファ#0 先頭アドレス（DATA）	バッファ#0の領域(256バイト）の先頭アドレスを指す。	
E9DB 〜DC	トラックNo. 格納アドレス（DATA）	最後にアクセスしたドライブのFAT のメッセージエリア内のトラックNo.を示すアドレス	
E9DD 〜DE	データポインタ（DATA）	バッファ#0内を指す。	
E9E0 〜E1	ドライブNo.	最後にLOADまたは、SAVEしたドライブNo.	
E9E2 〜EA	ファイル名	最後にLOADまたは、SAVEしたファイル名	

E 9 E B ～F 3	ファイル名	NAMEのときに使用	
E 9 F 4	DSKO$の トラックNo.		
E 9 F 5	DSKO$の セクタNo.		

ワークエリアー覧表

アドレス	機能・用途	備考
EA00 ～EB56	初期設定時に3A05～3B5Bの内容がここにブロック転送されてくる。	
EA00 ～02	OUT文で使用...... OUT(n),A	EA01…I/Oポートアドレス ACC…出力データ
EA03 ～10	割り算のルーチンよりCALL	
EA11	RND用カウンタ	
EA12	RND用ROMデータポインタ(1～3)	
EA13	RND用RAMデータポインタ(0～7)	
EA14 ～33	RND用RAMデータ。4バイトで1組(∵単精度データ) ここを変えると、乱数発生の系列が変わる。	
EA34 ～37	前回発生したRNDの値	RND(0)の値
EA38 ～3A	INP関数で使用...... IN A,(n)	EA39…I/Oポートアドレス ACC…入力データ
EA3B ～4E	USR関数のエントリーアドレス (USR0～USR9......2バイト1組) 未定義のUSR関数については、44A5Hが入っている。 (44A5へ飛ぶと、Illeagal function Callが出力される。)	EA48 TAB カウンタ
EA4F		
EA50 ～51	フリーエリア最終アドレスの初期値…E9FF (CLEAR文でも変化しない。)	EF54,55 (CLEAR文で変わる)
EA52 ～53	コマンド／ステートメント　中間言語テーブル開始アドレス	33BD
EA54	RS-232Cバッファオーバーフローフラグ	Overflow…00H
EA55	現在(割込み発生の直前)の割込みレベルが保存されている。 (Non-interrupt......FF)	
EA56	インタラプトがあったことを示すフラグ (interrupt...... not 0)	
EA57	00H以外を入れて、1文字出力ルーチンをCALLすると、 現在のカーソルのある位置の次の行をクリア。	

EA58	ターミナルモードのフラグ　　term □, □, □, □ bit 0 ::::…プリンタへのエコーバックをする／しない 1 ::::…コントロールコードを表示する／しない 2 ::::…ASC11／JIS 3 ::::…半二重／全二重(オートエコーする／しない) 4 ::::…パリティをつける／つけない 5 ::::…パリティ EVEN／ODD 6 ::::…CR コードでLFする／しない 7 ::::…ターミナルモードである／ない └── ターミナルモードのとき、このバイトが0になると困るので1にしてある。	01Hを入れたときのファンクションキーの機能 f·6 ：コントロールコードを表示する／しないのスイッチ f·8 ：ターミナルモードリセット f·9 ：画面プリント f·10：プリンタのラインフィード
EA59	カーソルのON／OFFをEA5FHに従わせよ！というフラグ	EA5FH…カーソルスイッチ
EA5A	ヌルキャラクタコード……カラー文の第2パラメータ、ホームクリアのとき画面に表示するキャラクタのアスキーコードを入れておく。……… 通常00H	ターミナルモード・01Hでf9にて画面プリントするとき、20Hを入れておけばスペースが無視されない。
EA5B	アトリビュートコード……… 通常00H リバース、アンダーライン、カラー指定などアトリビュートに入るもの 白黒モード　M｜－｜D｜P｜－｜R｜B｜S M：グラフィック／キャラクタ D：アンダーライン P：アッパーライン R：リバース B：ブリンク S：シークレット カラーモード　－｜－｜D｜P｜0｜R｜B｜S 　　　　　　　R｜G｜B｜M｜1｜－｜－｜－ 0 0 0 ……黒 0 0 1 ……青 0 1 0 ……緑 0 1 1 ……シアン(空) 1 0 0 ……赤 1 0 1 ……マゼンタ 1 1 0 ……黄 1 1 1 ……白	**カラーコード**  Y：黄 M：マゼンタ(桃) C：シアン(空) W：白 BL：黒
EA5C	ファンクションキー表示方法　50H…リバース 00H…ベースアトリビュート　画面をクリアして切換わる	
EA5D	スクロールの長さ……00Hで1行、01Hで2行……	
EA5E	スクロールの開始行＋1……01Hで最上行	
EA5F	カーソルスイッチ　　00H…OFF　00H以外…表示	locate文第3パラメータ
EA60	ファンクションキー表示スイッチ　00H…OFF 00H以外…表示	

<table>
<tr><th>アドレス</th><th>機能・用途</th><th>備考</th></tr>
<tr><td>EA61</td><td>カラー／白黒モード　00H……白黒　FFH……カラー</td><td></td></tr>
<tr><td>EA62</td><td>行数／白黒モード　14H……20行　19H……25行</td><td></td></tr>
<tr><td>EA63</td><td>カーソルのY方向の位置　＋1</td><td></td></tr>
<tr><td>EA64</td><td>カーソルのX方向の位置　＋1</td><td></td></tr>
<tr><td>EA65</td><td>白黒モードのときの1行の文字数　24H…36　48H…72
28H…40　50H…80</td><td></td></tr>
<tr><td>EA66</td><td>前にI/Oポート 30Hに出力したもの
<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td></td><td></td><td>BS2</td><td>BS1</td><td>MOTOR
ON／OFF</td><td>CINH</td><td>白黒／カラー</td><td>80／40</td></tr>
</table>
CINH：CMTインタフェースの出力するキャリア
のコントロール
0……2400Hz
1……1200Hz
BS1，2：CMT／SIOチャンネルの制御
BS2，BS1
0　0……CMTチャネル 600ボー
0　1……CMTチャネル 300ボー
1　0……SIOチャネル
1　1……禁止</td><td>ボー……ビット/秒
(転送速度の単位)</td></tr>
<tr><td>EA67</td><td>前にI/Oポート 40Hに出力したもの
<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td></td><td></td><td>BEEP</td><td></td><td>CLDS</td><td>CLK</td><td>CSTB</td><td>$\overline{\text{PSTB}}$</td></tr>
</table>
$\overline{\text{PSTB}}$……プリンタへのストローブ
CSTB……クロックに対するストローブ
CLK……クロックに対するシフトパルス
CLDS……CRTコントロール回路への同期
パルスを与える。</td><td>BEEP……ブザー</td></tr>
<tr><td>EA68</td><td>ファンクションキーフラグ......NOT ZEROでフラグON</td><td>ファンクションキーポインタ
EDC0，C1</td></tr>
<tr><td>EA69</td><td>キーボードから入力されたコード</td><td></td></tr>
<tr><td>EA6A</td><td>前回押した特殊キーのデータ</td><td>00……入力なし
01……シフトキー
02……コントロールキー
03……カナキー
04……カナ＋シフト
05……グラフキー</td></tr>
<tr><td>EA6B</td><td>リピートカウンタ（リピート0は 06H）</td><td></td></tr>
<tr><td>EA6C
～75</td><td>キーパターン</td><td></td></tr>
</table>

EA76 77 78	秒 分 } タイマリードをした時にセットされる。 時	コードは、BCDとなっていて、例えば10月は、 0AHではなく10H と入る。
79 7A 7B	日 月 } 初期値　79年1月1日 年	
EA7C ~1B	ファンクションキーの内容 1組16バイト（最後は00）で10組	3A81H～3B20に初期値が入っている。
EA7C～	f・1　H_T	
8C～	f・2　auto	
9C～	f・3　go to	
AC～	f・4　list	
BC～	f・5　run C_R	
CC～	f・6　time$	
DC～	f・7　key	
EC～	f・8　print	
FC～	f・9　list. C_R ↑ ↑	
EB0C	f・10　cont C_R	
EB1C ~29	PSET, PRESET, GET, PUT, LINEなどで使用	
EB2A ~45	RS-232C用	
EB46	ERRコード　　00：エラーなし	
EB47		
EB48	プリンタのヘッドの位置	LPOS関数の値
EB49	出力機器機番 (00…CRT　01～7F…プリンタ 80～FF…CMT)	
EB4A	スクリーンの実際の横の桁数	
EB4B		
EB4C		
EB4D	スクリーンへの表示　dis-able	
EB4E	PRINTルーチン（4742～）で使用	
EB4F		
EB50 ~51	スタックポインタの始まりアドレス （フリーエリアの最終から文字領域分を引いたところ）	
EB52 ~53	現在実行中の行番号	
EB54 ~55	プログラムテキスト開始番地 BASICプログラムスタート番地	16Kシステム　C021H 32Kシステム　8021H

アドレス	機能・用途	備考
EB56		
EB57 ~EC91	中間言語バッファ	
EC92 ~95		
EC96 ~ED93	キー入力バッファ（254バイト）	
ED94 ~98		
ED99	カーソルX座標	
ED9A ~B2	CONT CODE：その行が次の行とつながっているかどうかを示す。 ED9A……0行目　　00………つながっている 〜　　00以外…切れている EDB2……24行目	
EDB3	カセットのとき 00H	
EDB4 EDB5	X方向｝画面から文字を読むときの読みはじめ位置 Y方向｝（1行入力のときの初期カーソル位置）	
EDB6	どこまで読むか（X位置）	
EDB7 ~B8	TERMで使用	
EDB9	Disk で転送するセクタ数	
EDBA EDBB EDBC EDBD EDBE ~BF	アトリビュート要素 X方向｝グラフィック座標 Y方向｝ COLOR CODE 画面上のアドレス ｝（PSET，PRESET GET ，PUT LINE で使用）	
EDC0 ~C1	ファンクションキーポインタ ファンクションキー入力文字列の先頭アドレスを指す。	文字列最後は 00Hで判断
EDC2	キー入力でキーが押されていれば0になる。	
EDC3 ~C4	Y方向｝キャラクタ座標 X方向｝	EDBA~F 関連
EDC5 EDC6 EDC7	エラーカウンタ ドライブ#（0～　） ドライブの個数 ｝DISK	
EDC8		
EDC9	コントロールポートアドレス………DISK	
EDCA	LINEを引くキャラクタ	

EDCB	time up までの時間………DISK	
EDCC ~CD		
EDCE ~EE4D	RS-232C チャネル1バッファ（128バイト）	
EE4E ~EECD	RS-232C チャネル2バッファ（128バイト）	
EECE EED0	RS-232C 出力文字一時退避 RS-232C チャネル#（0 or 1）……出力時	
EED1 ~EF2B		
EF2C ~35	PUT／GET で使用	
EF36 ~3B	SAVE／LOAD のファイル名が入る（6バイト）	
EF3C	TAPEからの入力エラーがあったときに Tape read error として落とすかどうかのフラグ 00H……エラーにする	
EF3D	EF3C と同じような目的で設けられたと思うが使用していない。	
EF3E ~43	現在 LOAD 中のファイル名がはいる。	
EF44	変数認識ルーチンで、DIM 文から来たことを示すフラグ 00H以外	
EF45	FACC の型が入る。 02……整数型 03……文字型 04……単精度型 08……倍精度型	
EF46	中間言語⟷リスト　変換の両方のルーチンで使われる。中間言語の中だとか、DATA 文、REM 文の中だとかを示す。	
EF47	リスト⟶中間言語　ルーチンで数字を文番号として扱うことを示すフラグ	
EF48 ~49	ポインタの SAVE 場所	1単語を Get する。
EF4A	ACC の SAVE 場所	1単語を Get する。
EF4B	精度(02, 04または08)	1単語を Get する。
EF4C ~53	フローティングポイントレジスタ	1単語を Get する。

アドレス	機 能 ・ 用 途	備 考
EF54 ～55	フリーエリアの最終アドレス……CLEAR 文第2パラメータ	N-BasicのみのときE9FFHとなる。
EF56 ～57	ストリング(文字列)用スタックポインタ	
EF58 ～75	ストリングスタック（ディスクリプタのスタック） 3バイトで1組	1バイト　文字数 2バイト　アドレス
EF76 ～78	カレントストリングディスクリプタ（ストリングACC）	
EF79 ～7A	ストリングフリースペース頭	
EF7B ～7C		
EF7D ～7E	ガベージコレクション中でHL Save（変数ポインタ）	
EF7F ～80	For 文中でHL Save（テキストポインタ）	
EF81 ～82	Read 文ポインタがある行の行番号	
EF83	変数認識で使用。どこから呼ばれたかを示す。	
EF84	Read であることを示すフラグ	
EF85 ～86		
EF87	行番号──→アドレスが起こったことを示すフラグ	
EF88	AUTO実行中を示すフラグ	
EF89 ～8A	今度発生する行番号	
EF8B ～8C	行番号間かく。	
EF8D ～8E	1文を実行する前のHLの値	
EF8F ～90	1文を実行する前のSPの値	
EF91 ～92	ERL（エラーの起った行の行番号）	
EF93 ～94	ERL コピー（プログラム実行中だったとき）	BREAK時の行番号
EF95 ～96	ERROR 文のアドレス	

EF97 ~98	ON ERROR GOTO～の行先アドレス	
EF99	ON ERROR GOTOのフラグ	
EF9A ~9B	HL SAVE	
EF9C ~9D	ERLコピー	
EF9E ~9F	ERROR文のアドレス	
EFA0 ~A1	変数領域の始まりアドレス	このアドレスの前2バイトが、プログラムエンドを意味するリンクポインタ00H，00Hとなっている。
EFA2 ~A3	配列変数領域の始まりアドレス	
EFA4 ~A5	フリーエリアの先頭アドレス	
EFA6 ~A7	READ文ポインタ （現在DATA文中READしている部分のアドレス）	
EFA8 ~C1	変数の暗黙型…DEF文で変わる。（変数の型指定） （A～Z　26文字分）	
EFC2 ~F090	FN関数で使用 引数のテーブルなどに使用する。	
F091	不明	
F092 ~95	変数のサーチで使用	
F096 ~97	ガベージコレクションで使用	
F098 ~99	FNで使用	
F09A ~A1	SWAP用フローティングポインタレジスタ	
F0A2	TRONフラグ……01HでON	

アドレス	機能・用途	備考
F0A3 ~AB	浮動小数点アキュムレータ(USR関数引数) F0A3 A4 A5 A6 A7 A8 ┐ 整数型 (02) (A8～A9) A9 ┘ AA AB 単精度型 (04) (A8～AB) 倍精度型 (08) (A4～AB)	EF45Hに型が入る。
F0AC	符号	
F0AD ~B5	倍精度演算用数値　F0AD ~ F0B5…符号	
F0B6 ~BC	HEX$, OCT$用バッファ	
F0B7 ~CD	F Acc→ストリング用バッファ	
F0CE ~D0	空き (上のバッファに含まれるが、実際には、使用されない。)	
F0D1 ~D8	倍精度割り算で使用	
F0D9 ~DD		
F0DE ~E0	単精度　乗算で使用	
F0E1 ~F215	N-BASIC(DISK BASIC)用ジャンプテーブル 3バイト1組で100組	詳細は、次頁からのジャンプテーブル参照。
F216 ~FF	空き	
F300 ~FEB7	ビデオラム(3Kバイト) 1行120バイト(80バイト画面、40バイト…アトリビュート)	詳細は画面表示の項参照。
FEB8 ~FF2F	クリアするときビデオRAMに入れるもの1行分 1行120バイト(80バイト画面、40バイト…アトリビュート)	
FF30 ~3C	モニタで使用	
FF3D ~FF	イニシャライズ後は、空き	

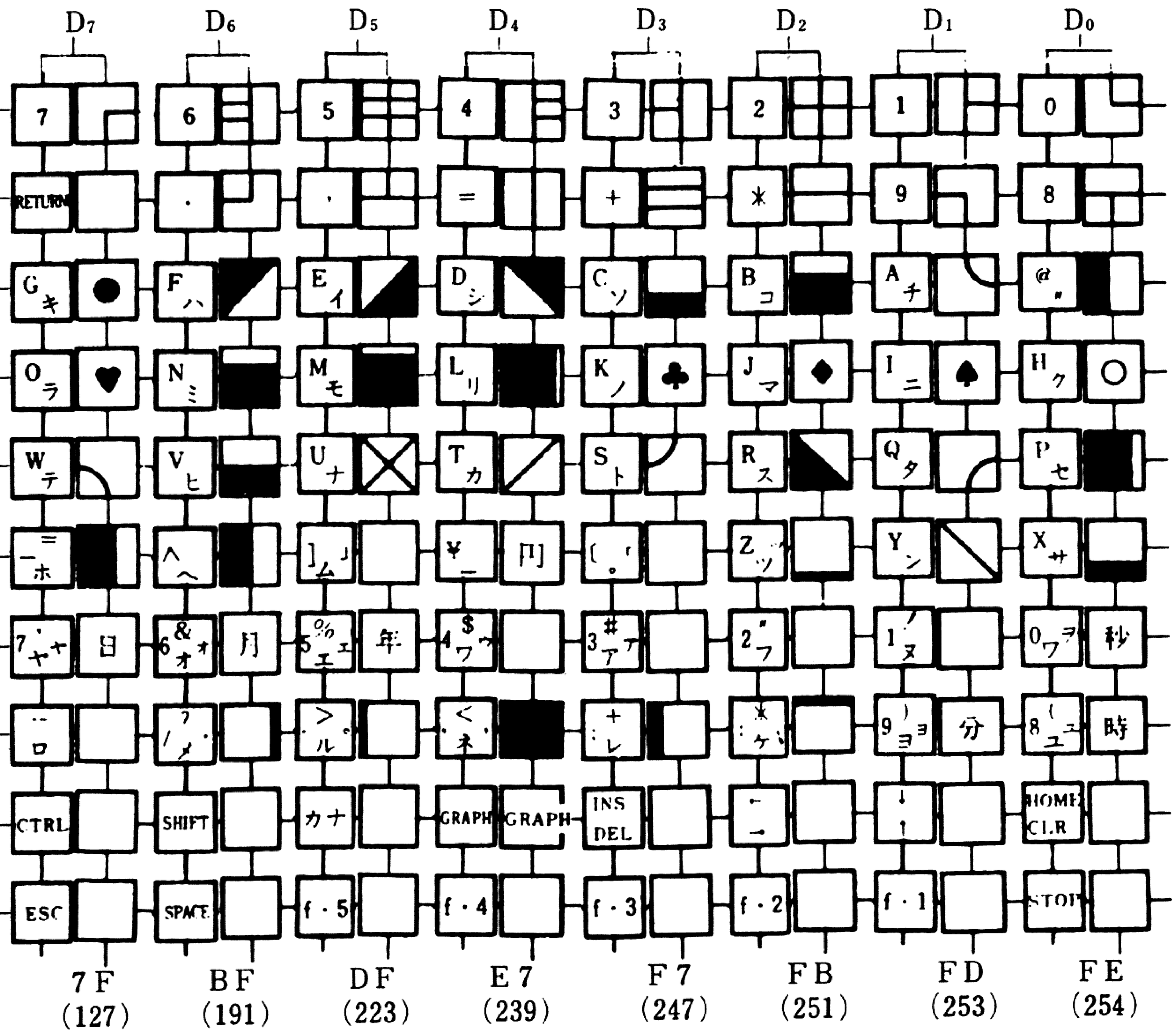
ジャンプテーブル一覧表

アドレス	N-BASIC	Disk BASIC	備　　　考
F0E1	1875	E1EB	〈CVI〉
E4	〃	E1EE	〈CVS〉
E7	〃	E1F1	〈CVD〉
EA	〃	E84C	〈EOF〉
ED	〃	E823	〈LOC〉
F0F0	〃	E83B	〈LOF〉
F3	〃	E1D2	〈MKI$〉
F6	〃	E1D5	〈MKS$〉
F9	〃	E1D8	〈MKD$〉
FC	〃	1875	〔CMD〕
FF	〃	DBD6	〔OPEN〕
F102	〃	E0BF	〔FIELD〕
5	〃	1875	〔ISET〕
8	〃	1875	〔IRESET〕
B	〃	1875	〔TALK〕
E	〃	1875	〔LISTEN〕
F111	〃	1875	〔MAT〕
4	〃	1875	〔POLL〕
7	〃	1875	〔WBYTE〕
A	〃	1875	〔RBYTE〕
D	〃	1875	?
F120	〃	1875	〈BCD$〉
3	〃	1875	〈DEC〉
6	〃	E27F	〔LFILES〕
9	〃	E327	〔GET ＃〕 from 1888H
C	〃	E47C	〔DSKO$〕
F	〃	D804	〔FORMAT〕
F132	〃	E326	〔PUT ＃〕 from 1893H
5	〃	E240	〔CLOSE〕
8	〃	DDF0	〔LOAD〕
B	〃	DDF1	〔MERGE〕
E	〃	DB84	〔NAME〕
F141	〃	DDDE	〔KILL〕
4	〃	E012	〔LSET〕
7	〃	E011	〔RSET〕
A	〃	DE3D	〔SAVE〕
D	〃	E284	〔FILES〕
F150	〃	E20D	〔REMOVE〕
3	〃	E208	〔MOUNT〕
6	〃	E109	〔SET〕
9	〃	E812	〈DSKF〉
C	〃	E44A	〈DSKI$〉
F	〃	E15D	〈ATTR$〉
F162	〃	DB60	from 2327H
5	〃	E441	from 2345H
8	〃	E868	〈FPOS〉
B	0000	D70A	from F16BH ?　エラーメッセージ出力、データアドレスセット後
E		E896	from 3C8CH　コマンド待ちに戻る前
F171		——	未使用
4		E871	from 3D0EH　コマンド待ちのとき、何かを入力、RETを押したあと

アドレス	N-BASIC	Disk BASIC	備　　考
F177	0000	D6CC	from　3D6AH
A	〃	D6D3	〃　3D70H
D	〃	D766	〃　3E23H, 4331H
F180	FFFF	D6DA	〃　4873H…プリンタ処理後の後始末
3	〃	D6E2	〃　40A7H…RST18で通る。
6	〃	——	〃　413FH
9	〃	D6C0	〃　4540H "RUN"において、あたまに何かがついたとき
C	〃	D774	〃　4745H "PRINT"のあたま
F	〃	D700	〃　4791H "PRINT"の途中
F192	〃	——	〃　47EAH
5	〃	D71A	〃　47EFH, 4838H
8	〃	D737	〃　48E2H "INPUT"で%でなかったとき
B	〃	D82B	〃　497AH
E	〃	——	〃　0F9FH キーボートセンス
F1A1	〃	E8AC	〃　4885H "LINE INPUT #"のときCALL
4	〃	——	〃　0C88H カセットI/O…IN
7	〃	D780	〃　0175H
A	〃	D6EE	不明
D	〃	D76F	from　3C0DH
F1B0	〃	——	〃　0081H Stop-RESETしたとき通る。
3	〃	——	〃　009CH ディスクI/O
6	〃	——	〃　0BF3H カセットI/O…READ
9	〃	——	〃　0C46H カセットI/O…WRITE
C	〃	——	〃　0D61H 1文字プリンタ出力
F	0000	——	〃　0DB8H "TERM"の頭
F1C2	〃	——	〃　4BF9H 次にF1FEHをCALL
5	〃	——	〃　4E47H "DIM"コマンドの始めの方
8	〃	D79E	〃　3BF9H エラー処理ルーチンの頭
B	〃	——	〃　3EA8H
E	〃	——	〃　5818H
F1D1	〃	——	〃　547CH
4	〃	——	〃　5306H
7	〃	——	〃　16C5H
A	〃	——	〈RST 20H〉
D	〃	——	〈 〃 28H〉
F1E0	〃	——	〈 〃 30H〉
3	〃	——	〈 〃 38H〉
6	〃	——	未使用？
9	〃	——	
C	〃	——	
F	〃	——	
F1F2	〃	——	from　4742H "PRINT"、"LPRINT"の頭
5	〃	——	〃　48DAH
8	〃	——	〃　4847H "LINE"の頭
B	〃	——	〃　4939H "READ"の頭
E	00FF	——	〃　4BFCH
F201	FFFF	——	〃　4642H
4	〃	——	〃　41FAH
7	〃	——	〃　16C2H
A	〃	——	〃　0E37H IN
D	〃	——	〃　0CDAH 8251関係、カセットI/O…OUT
F210	〃	D74E	?
3	〃	——	未使用

I/Oポート一覧表

0～9（入力）KEY BOAD



ポートアドレス		内容
0	0H	KEY BOAD（上図）
1	1H	
2	2H	
3	3H	
4	4H	
5	5H	
6	6H	
7	7H	
8	8H	
9	9H	
10～15	AH～FH	未使用
16	10H	プリンタ出力用 μPD1990用データ （NEC, μPD1990データシート参照のこと。） 注）関連ポートとして40H参照のこと。 （プリンタストローブ、READY, μPD1990コントロールその他）
17～31	11H～1FH	ポート10Hのイメージ

ポート アドレス		内　　　容
32	20H	（入出力） μPD8251（USART）データワード
33	21H	（入出力） μPD8251（USART）コントロールワード （NEC μPD8251データシート参照のこと。） ↑プログラマブル・コミュニケーション・インターフェース （Universal Synchronous / A synchronous Receiver / Transmiter）
34 〜 47	22H 〜 2FH	20H，21Hのイメージ
48	30H	（出力）

	MSB ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	LSB ビット0
出力	未使用	未使用	BS2	BS1	ON…1 MOTOR OFF…0	CINH	1 ↑ 白黒　カラー ↓ 0	1 ↑ 80字行　40字行 ↓ 0

ワークエリア（EA66H）参照

49 〜 63	31H 〜 3FH	30Hのイメージ
64	40H	（入出力）

	MSB ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	LSB ビット0
出力	未使用	未使用	BEEP	未接続	CLDS CLK	μPD1990 データ・ シフト・パルス	μPD1990 コントロール ・パルス	プリンタ $\overline{\text{ストローブ}}$
入力	未使用	未使用	DMA $\overline{\text{BUSY}}$	μPD1990の 出力データ	$\overline{\text{EXTON}}$	μPD8251の 入力		プリンタ $\overline{\text{READY}}$

ワークエリア（EA67H）参照

65 〜 79	41H 〜 4FH	40Hのイメージ
80	50H	出力 μPD3301（CRTC） データ・ポート
81	51H	出力 μPD3301（CRTC） コントロール・ポート
96	60H 〜	μPD8257（DMAC）用
100	64H	（出力） 8257のDMA Ch2アドレス
101	65H	（出力） 8257のCh2ターミナルカウント

104	68H	（出力）8257のモードセット　レジスタ 00H………DMA　OFF
112 〜 127	70 〜 7F	未使用
128 〜 175	80 〜 AF	拡張I/O(8011,12)にて使用可
176 〜 179	B0 〜 B3	汎用パラレルI/Oポート（8011） B0……入力（8 bit） B1……出力（8 bit） B2……入力（4 bit） B3……出力（4 bit）
180 〜 191	B4 〜 BF	未使用
192	C0	チャネル1データ　｝8251（RS-232C）
193	C1	チャネル1コマンド指定　｝8251（RS-232C）
194	C2	チャネル2データ　｝8251（RS-232C）
195	C3	チャネル2コマンド指定　｝8251（RS-232C）
196 〜 223	C4 〜 DF	未使用
224 〜 227	E0 〜 E3	（出力） 拡張ユニットモード設定
228	E4	（出力） ? Out（E4H）A　008Bで実行されている。
229 230	E5 E6	未使用
231	E7	（出力） 拡張ユニットROM, RAM切換コントロール
232 〜 238	E8 〜 EE	未使用
239	EF	（出力） FGU-8000(高解像フルグラフィックユニット） VRAM切換コントロール
240 〜 251	F0 〜 FB	未使用

ポート アドレス		内容
252 ～ 255	FC ～ FF	フロッピーディスク FC（8 bit）…………入力 FD（8 bit）…………出力 FE（上位 4 bit）……入力 （下位 4 bit）……出力 FF μPD8255モードセット └ フロッピーディスク用PIO （8 ビット）……出力

5 KEYWORD, 中間言語コードと処理ルーチン対応表

```
100 CLEAR 2000:DIM AD$(130):K=1:AD$(0)=""
110 FOR AD=&H33BD TO &H34BF STEP 2
120 A=PEEK(AD)+PEEK(AD+1)*256
130 AD$(K)=RIGHT$("000"+HEX$(A),4)+" "
140 IF A<&H7FFF THEN 160
150 A=PEEK(A+1)+PEEK(A+2)*256:AD$(K)=AD$(K)+RIGHT$("000"+HEX$(A),4)
160 K=K+1:NEXT
170 A=ASC("A"):AD=&H34F5
180 KW$=CHR$(A)
190 KW=PEEK(AD):IF KW=0 THEN A=A+1:AD=AD+1:GOTO 180
200 IF KW<&H80 THEN KW$=KW$+CHR$(KW):AD=AD+1:GOTO 190
210 KW$=KW$+CHR$(KW-&H80):AD=AD+1:C=PEEK(AD)
220 IFC<&H80 THEN C$="FF "+HEX$(C+&H80):AC$="･"+CHR$(C+&H80):GOTO250
230 C$=HEX$(C):IF C<&HF8 THEN AC$=CHR$(C) ELSE AC$="･"
240 IF ASC(KW$)=&H5B THEN KW$=MID$(KW$,2)
250 IF C<&H2E THEN K=C+&H55:GOTO 270
260 IF C>&H80 AND C<&HD6 THEN K=C-&H80 ELSE K=0
270 LPRINT KW$TAB(8)C$TAB(15)AC$TAB(19)AD$(K)
280 AD=AD+1:IF AD<&H37BB THEN 180
```

キーワード	中間言語コード	アスキーコード	処理ルーチン
AND	F8	･	
ABS	FF 86	･■	2671
ATN	FF 8E	･▌	3372
ASC	FF 95	･─	5498
AUTO	A9	ｩ	46CF
ATTR$	E7	◤	
BCD$	FF 9E	･╰	F120 1875
BEEP	B2	ｲ	0D41
CONSOLE	9F	╯	0884
CLOSE	CA	ﾊ	F135 E240
CONT	99	┐	4383
CLEAR	92	┤	44E8
CLOAD	9B	┘	1F10
CSAVE	9A	└	1EC0
CSRLIN	E6	◥	
CINT	FF 9F	･╯	277F
CSNG	FF A0	･	27B3
CDBL	FF A1	･｡	270F
CVI	FF A3	･｣	F0E1 E1EB
CVS	FF A4	･､	F0E4 E1EE
CVD	FF A5	･･	F0E7 E1F1
COS	FF 8C	･▌	32F6
CHR$	FF 96	･│	54A8
CMD	B8	ｸ	F0FC 1875
COLOR	B5	ｵ	0951
DATA	84	▄	458E
DIM	86	■	4E37
DEFSTR	AB	ｫ	445B
DEFINT	AC	ｬ	445E
DEFSNG	AD	ｭ	4461
DEFDBL	AE	ｮ	4464
DSKO$	C2	ﾂ	F12C E47C
DEF	97	│	50CC
DELETE	A8	ｨ	58D9
DSKI$	E8	♠	
DSKF	FF A6	･ｦ	F159 E812
DEC	FF 9D	･╮	F123 1875
DATE$	EB	♣	
END	81	▁	432F
ELSE	A1	｡	45C0
ERASE	A5	･	43DF
ERROR	A6	ｦ	46C4
ERL	DF	ﾟ	
ERR	E0	═	
EXP	FF 8B	･▌	31F3
EOF	FF A7	･ｧ	F0EA E84C
EQV	FB	･	
FORMAT	B3	ｳ	F12F D804
FOR	82	▂	4159
FIELD	C6	ﾆ	F102 E0BF
FILES	CD	ﾍ	F14D E284
FN	DC	ﾜ	
FRE	FF 8F	･┼	5051
FIX	FF A2	･｢	282C
FPOS	FF AA	･ｪ	F168 E868
GOTO	89	▎	456D
GO TO	89	▎	456D
GOSUB	8D	▊	4555

GET	C7	ﾇ	1886	
HEX$	FF 9A	・└	5275	
INPUT	85	■	48DA	
IF	8B	▌	4702	
INSTR	E3	┤		
INT	FF 85	・■	283F	
INP	FF 90	・┴	56A1	
IMP	FC	・		
INIT	D4	ﾔ	2262	
INKEY$	E9	♥		
ISET	BD	ｽ	F105	1875
IRESET	BE	ｾ	F108	1875
IEEE	FF EC	・●		
KILL	CF	ﾏ	F141	DDDE
KEY	B4	ｴ	1343	
LET	88	│	45DE	
LOCATE	D5	ﾕ	0792	
LINE	AF	ｯ	4877	
LOAD	CB	ﾋ	F138	DDF0
LSET	D0	ﾐ	F144	E012
LPRINT	9D	┐	473A	
LLIST	9E	└	5707	
LPOS	FF 9B	・┘	5074	
LISTEN	C1	ﾁ	F10E	1875
LIST	93	├	570C	
LFILES	D3	ﾓ	F126	E27F
LOG	FF 8A	・▌	2503	
LOC	FF A8	・ｨ	F0ED	E823
LEN	FF 92	・┤	548C	
LEFT$	FF 81	・▁	54F9	
LOF	FF A9	・ｩ	F0F0	E83B
MOUNT	C4	ﾄ	F153	E208
MERGE	CC	ﾌ	F13B	DDF1
MOD	FD	・		
MKI$	FF AB	・ｫ	F0F3	E1D2
MKS$	FF AC	・ｬ	F0F6	E1D5
MKD$	FF AD	・ｭ	F0F9	E1D8
MID$	FF 83	・▃	5532	
MOTOR	B9	ｹ	0DA1	
MON	B7	ｷ	0D5D	
MAT	C0	ﾀ	F111	1875
NEXT	83	▃	4A08	
NAME	CE	ﾎ	F13E	DB84
NEW	94	▔	3DE0	
NOT	DE	ﾞ		
OUT	9C	╭	56AD	
ON	95	─	4642	
OPEN	C5	ﾅ	F0FF	DBD6
OR	F9	∴		
OCT$	FF 99	・┐	5270	
PUT	C8	ﾈ	1891	
POKE	98	┌	5918	
PRINT	91	┬	4742	
POS	FF 91	・┬	5079	
PEEK	FF 97	・│	5911	
PORT	FF 9C	・╭	20F9	
POLL	BA	ｺ	F114	1875
PSET	B1	ｱ	06B8	

PRESET	B0	ｰ	0705	
POINT	EF	＼		
READ	87	█	4939	
RUN	8A	▌	453D	
RESTORE	8C	▌	4302	
RETURN	8E	▌	45A3	
REMOVE	C3	ﾃ	F150	E20D
REM	8F	┼	45C0	
RESUME	A7	ｧ	468C	
RSET	D1	ﾑ	F147	E011
RIGHT$	FF 82	・▂	5529	
RND	FF 88	・│	3283	
RENUM	AA	ｪ	5AED	
RBYTE	BB	ｻ	F11A	1875
STOP	90	┴	432A	
SWAP	A4	､	439C	
SET	C9	ﾉ	F156	E109
SAVE	D2	ﾒ	F14A	DE3D
SPC(	DD	ﾝ		
STEP	DA	ﾚ		
SGN	FF 84	・▄	2686	
SQR	FF 87	・█	31A1	
SIN	FF 89	・▌	32FC	
STR$	FF 93	・├	527A	
STRING$	E1	╞		
SPACE$	FF 98	・┌	54DF	
STATUS	EE	／		
SRQ	ED	○		
TRON	A2	｢	4396	
TROFF	A3	｣	4397	
TAB(	D9	ﾙ		
TO	D7	ﾗ		
THEN	D8	ﾘ		
TAN	FF 8D	・▌	335D	
TERM	B6	ｶ	0DB8	
TALK	BF	ｿ	F10B	1875
TIME$	EA	◆		
USING	E2	╪		
USR	DB	ﾛ		
VAL	FF 94	・▔	5553	
VARPTR	E5	◣		
WIDTH	A0		0843	
WAIT	96	│	56B3	
WBYTE	BC	ｼ	F117	1875
XOR	FA	・		
+	F3	月		
-	F4	日		
*	F5	時		
/	F6	分		
^	F7	秒		
¥	FE	・		
'	E4	◢		
>	F0	×		
=	F1	円		
<	F2	年		

6 演算順位表

1. (　)でくくられた式
2. べき乗　A＾2……A^2
3. 負数
4. ＊１
5. ￥
6. mod
7. ＋－
8. 条件判定
9. NOT
10. AND
11. OR
12. XOR
13. IMP
14. EQV

演算は、同順位の場合、左から右へ順に行なわれます。

7 キャラクタ・コード表(ASCIIコード表)

上位4ビット→

下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		
1	S_H	D_1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	C_R	←	-	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	_	o				ッ	ソ	マ	゜		

注1) ASCII……American Standard Code for Information Interchange
(アスキー)

注2) 7BH～7EH, 94H, 97Hは対応するキーがありませんので、画面やプリンタなどに出力する場合は、CHR$関数を用いる必要があります。
print chr$(&H7B)……プリンタの場合lprint chr$(&H7B)

注3) コントロールコード部分(00H～1FH)については、コントロールコードの項をご覧ください。

注：この付録に記載されているN-BASIC・Disk BASICインタプリタおよびワークエリア等のアドレス・機能は、システムソフトが独自に解析したものであり、運用上の影響については、責任を負いかねますのでご了承ください。

索引

〈著者代表略歴〉

牟田　慎一郎

昭和20年1月27日生まれ

昭和43年　九州工業大学電気工学科卒業

現在　九州松下電器(株) 勤務

本書の内容に関しまして、ご質問・ご意見がございましたら下記の出版部まで書面にてお寄せくださいますようお願い致します。

PCファミリー・テクニカル・ノウハウ集

PC-8000シリーズ編

PC-Tech Know 8000（テクノウ）　©1982 システムソフト

定価 2,900円

昭和57年　4月1日　初版発行

昭和57年　5月1日　第2刷発行

共　著　牟田慎一郎　田川嘉隆　平松達雄　栗山浩一

監　修　システム　ソフト

発行者　樺島正博

発行所　福岡市中央区天神2丁目14-8 福岡天神センタービルB1（〒810）

株式会社システムソフト福岡　電話092・714・6254

印刷・製本　秀巧社印刷株式会社(落丁・乱丁本はお取替いたします。)

表紙イラスト　中山　昭次

「BASICホームコンピュータプログラム集」
チャールズ・D・スタンバーグ著
藤　田　英　時訳
B5判、312ページ、定価2,900円

マイコンの初心者からエキスパートまで、すぐに役立つ実用プログラムを約80種類、収容。プログラムは内容、機能、操作法、完全リスト、使用変数名、サンプル、データ、サンプル・ラン、出力結果がそれぞれ詳細に解説され、そのまま利用できるほか、簡単に変更・拡張が可能です。また、テープやディスク等の外部記憶装置を使わなくても実行できるのが特徴です。

〈主要目次〉
- **家庭財政プログラム**
- **自動車関連プログラム**
- **キッチン・プログラム**
- **家庭のスケジュール管理**
- **多目的検索プログラム**
- **家庭教育・学習プログラム**
- **単位交換プログラム**
- **レクリエーション・プログラム**
- **ホビイスト日誌**

「8086入門」
ステファン・P・モールス著
内　藤　祥　雄訳
A5判、218ページ、定価2,900円

16ビット・プロセッサ8086のアーキテクチャ、システム設計、プログラミング法を初めて紹介。
著者は8086プロセッサの設計者で、8086のすべてを網羅。
マイクロプロセッサの説明から始まり、8086の構造、システムの基本設計、さらにアセンブラ言語および高級言語によるプログラミング法を解説。豊富な例とイラストでわかりやすく説明されているため、初心者、専門家ともに利用価値大。

〈主要目次〉
- **8086のアーキテクチャ**
- **8086の命令セット**
- **8086のシステム設計**
- **8086のアセンブリ言語**
- **8086の高級言語**

発刊予定

「BASICビジネスプログラム集」
B5判、予価2,900円

「BASICホームコンピュータプログラム集」に続く本格的ビジネス版！パソコンで事務処理を行なうための実用的ビジネス・アプリケーションを約35種類掲載。

〈主要目次〉
第１部　財務管理と財務分析
- **簡易簿記システム**
- **受取勘定システム**
- **財務プログラム**

第２部　在庫管理と在庫分析
- **汎用在庫システム**
- **定期在庫システム**
- **在庫プログラム**

第３部　生産計画と生産管理
- **生産プログラム**
- **製造原価計算**

SYSTEM SOFT